Joint Server Assignment and Resource Management for Edge-based MAR System

Can Wang, Sheng Zhang, *Member, IEEE,* Zhuzhong Qian, *Member, IEEE,* Mingjun Xiao, *Member, IEEE,* Jie Wu, *Fellow, IEEE,* Baoliu Ye, *Member, IEEE,* and Sanglu Lu, *Member, IEEE*

Abstract—Mobile Augmented Reality (MAR) applications usually contain computation-intensive tasks which far outstrip the capability of mobile devices. One way to overcome this is offloading computation-intensive MAR tasks to remote clouds. However, the wide area network delay is hard to reduce. Thanks to edge computing, we can offload MAR tasks to nearby servers. Prior studies focus on either single-task MAR applications offloading or dependent tasks offloading for a single user. In this paper, we study the offloading decision of MAR applications from multiple users, each of which is comprised of a chain of dependent tasks, over a generic cloud-edge system consisting of a group of heterogeneous edge servers and remote clouds. We formulate the Multi-user Multi-task MAR Application Scheduling (M^{3}AS) problem, which is NP-hard. We present Mutas, an efficient scheduling algorithm that jointly optimizes server assignment and resource management. We also consider the online version of M^{3}AS and present OnMutas. Extensive evaluations demonstrate that both Mutas and OnMutas can significantly reduce the service delays of MAR applications when compared to three other heuristics.

Index Terms—mobile augmented reality, edge computing, multi-task application, scheduling, resource management

1 Introduction

Mobile augmented reality (MAR) enhances our perception of the surrounding physical environment by mixing the real world with computer generated visual information [1]. Due to the wide spread of mobile devices together with advances on computer vision, MAR applications are gaining popularity in various industries [2], such as navigation, tourism, entertainment and so on. The MAR market is predicted to be worth $108 billion by 2021 [3].

MAR applications generally contain complicated deep-learning algorithms as core components; however, running these algorithms in mobile devices raises critical challenges, i.e., high processing delay and power consumption [4]. One way to overcome these limitations is to use cloud resources for the execution of computation intensive tasks. However, compared to the ever increasing data processing speed, the wide area network bandwidth has come to a standstill [5], thus cloud-based MAR may incur excessive transmission delay.

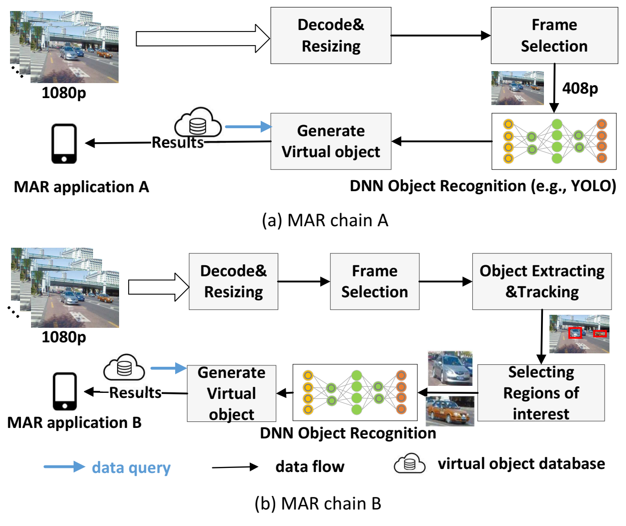
Edge computing is a new computing paradigm which advocates processing data at the edge of the network [6]. Leveraging edge computing, the computation intensive tasks in MAR applications can be offloaded to nearby edge servers. Since edge servers are placed in proximity to users, the problem of high network delay is mitigated.

Indeed, more and more MAR applications can be modeled as chains of dependent tasks [7], each of which represents a certain operation on the input messages (e.g., video frames) and passes its outputs down to the succeeding task. Fig. 1 shows two example MAR applications, both of which are constructed as pipelines of tasks. Different MAR applications may consist of different numbers of tasks. For example, in general MAR applications [1], video analysis results could be derived from the direct object recognition task, while some sophisticated MAR applications may involve additional operations (e.g., object tracking task).

In this paper, we consider the offloading of multi-task MAR applications from multiple users, over a generic cloud-edge computing system comprising of a group of heterogeneous edge servers and remote clouds. We make task offloading decisions for all mobile devices, based on considerations such as the network latency, the amount of computations and the resource requirement. This is a server assignment problem. On the other hand, a remote cloud or edge server has to determine how to distribute computing resources among all the tasks assigned to it, which is a resource management problem.

There are many studies concerning with the application scheduling in the edge environment, however, they usually consider applications with two components, one running in users' devices and the other running in edge servers [8], [9], [10]. Only a few works consider the offloading of multiple dependent tasks [11], [12], [13], [14]. Despite the works that have been done, they have some inappropriate assumptions such as negligible

- *C. Wang, S. Zhang, Z.Z. Qian, B.L. Ye, and S.L. Lu are with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, 210023, China.*
  E-mail: MF1733063@smail.nju.edu.cn, {sheng, qzz, yebl, sanglu}@nju.edu.cn.
- *M.J. Xiao is with with the School of Computer Science and Technology / Suzhou Institute for Advanced Study, University of Science and Technology of China, Hefei, China.*
  E-mail: xiaomj@ustc.edu.cn.
- *J. Wu is with the Center for Networked Computing, Temple University, Philadelphia, PA 19122, USA.*
  E-mail: jiewu@temple.edu.



(a) MAR chain A

(b) MAR chain B

Fig. 1. Example of AR applications. Different MAR applications may consist of different numbers of tasks.

data communication delay and machine-irrelated task processing time. Most importantly, these existing designs either assume unlimited computing resources on edge servers or consider only the single-user scenario. To the best of our knowledge, most existing related works are not practical for multi-user MAR systems where resource contention exits and resource management is required.

Challenges and Solution. There major challenges for designing an effective scheduler for the edge-based multi-user multi-task MAR system can be summarized as follows: 1) The distribution of mobile users exhibits highly spatial diversity [15]. Such a user distribution will lead to imbalanced workloads among edge servers, and thus a scheduler which can dynamically dispatch MAR tasks is needed. 2) The precedence constraints and data transfer requirements between tasks can drastically complicate the scheduling decision [16], [17], [18], [19]. 3) Edge servers are usually less powerful than cloud servers, hence resource management becomes crucial. 4) Tasks from multiple users may be diverse in data size and computation complexity, being agnostic to task type and data size makes simple fair sharing far from ideal for multi-user multi-task MAR systems.

To address the challenges above, we design a scheduler to dynamically dispatch MAR applications to an edge-cloud network, and also allocate resources to each of them. In this paper, a MAR application is modeled as a chain of inter-dependent tasks with data transmissions between them, where each task can be offloaded to different servers. The objective is to identify a server assignment and resource allocation solution that minimizes the average service delay for all MAR users.

Contributions. To the best of our knowledge, our work is the first to decompose MAR applications as pipelines of tasks in an edge environment, and design both offline and online scheduling algorithms for a multi-user MAR system. Our main contributions are as follows:

- We formulate the Muti-user Multi-task MAR Application Scheduling (M^{3}AS) problem where various MAR applications with dependent tasks are scheduled over an edge-cloud computing system for delay minimization.
- We analyze the M^{3}AS problem through transformation and relaxation, then we design a scheduling algorithm named Mutas for the batch processing of MAR applications, which boosts the performance of an edge-based MAR system by jointly optimizing server assignment and resource allocation.
- We also discuss the online version of the M^{3}AS problem. We present OnMutas, which adaptively places MAR tasks among different servers and decides how much computational capacity is provisioned for each task.
- We conducted extensive experiments to evaluate the performance of Mutas and OnMutas. The experimental results show that both our offline and online algorithm can significantly reduce the service delays of MAR applications when compared to three other heuristics. Our evaluation results also confirm that our MAR system can support the integration of various MAR applications.

The rest of this paper is organized as follows. Section 2 reviews the related work. Section 3 describes the system model and the problem formulation. Section 4 develops the Mutas algorithm. Section 5 studies the online version of our problem and Section 6 evaluates our proposed design using simulations. We present our system architecture and detail the experiments in Section 7. Some extensions are discussed in Section 8 and conclusions are made in Section 9.

2 Related Work

Earlier work on application placement and scheduling in Mobile Edge Computing (MEC) has considered applications with two components, one running in a cloud (can be either edge cloud or core cloud) and the other running in a user's device [8], [9]. For example, in [10], a dynamic programming algorithm is proposed to handle deterministic and stochastic application deadlines. However, it only considers offloading from a mobile device to a remote cloud. Moreover, they improperly assume that the mobile device has infinite capacity and can execute any number of tasks simultaneously without impacting the processing time of each task. This assumption is unrealistic since user devices are usually resource limited, while we focus on multi-task MAR applications that can be deployed and executed across one or multiple edge clouds. The offloading of dependent tasks is complicated by the need to satisfy precedence constraints.

The problem of offloading dependent tasks to multiple types of edge servers has been considered in [16], [20], and [21]. In [16], a fully polynomial time approximation scheme is proposed to minimize the overall delay under a resource cost constraint. However, the devices are again assumed to possess infinite capacity in terms of the number of tasks that can be processed simultaneously without reduction in the processing speed for each task. In [22], the online placement of multi-task applications in edge environments is investigated, in which they jointly consider task assignment and bandwidth allocation. However, these load balancing schemes are mainly designed for tree edge-cloud networks and the main objective is to optimize the network delay while the processing time of each task is fixed.

In [19], the authors build a more generalized processing model, and a load-balancing heuristic algorithm is proposed for makespan minimization with dependent tasks. However, they do not consider resource sharing while the proposed scheduling algorithm must have knowledge of the execution times of all tasks on different servers. In addition, their design only aims to solve the placement of a single application; more applications cannot be processed simultaneously.

On the contrary, in this paper, we aim to design a scheduler for a multi-user MAR system. Resource management and task dispatching are jointly considered; we assume that the computing resource on an edge server is limited and may be shared by all the tasks assigned to it. This leads to a unique problem formulation that has not been considered in the existing literature.

3 System model

3.1 Generalized edge-cloud network

We consider an edge-cloud network with a set of N servers, denoted by $\mathcal{N} = \{s_1, s_2, ..., s_N\}$, including edge servers and cloud servers. Here edge servers are heterogenous and may be installed in edge computing hosts, cloudlet devices, or peer mobile devices [19]. As we mentioned before, edge servers may be diverse in physical locations and computing capability, connected with each other through the Internet backbone. The network architecture is illustrated in Fig. 2.

We further assume several remote cloud centers. Compared with edge servers, a cloud center has richer computing resources; however, application offloading to the remote cloud suffers from long delays between the mobile devices and the remote cloud. Let $d_{i,j}$ be the end to end delay between server s_i and s_j. For simplicity of illustration, we assume $d_{i,j} = d_{j,i}$ and $d_{i,j} = 0$ if $i = j$.

3.2 MAR application model

There are a set of M users in the MAR system, denoted by $\mathcal{U} = \{u_1, u_2, ..., u_M\}$. Each user has an AR application to be executed. As we mentioned before, an AR application includes many computation-intensive computer vision and artificial intelligence algorithms [4] as key components (e.g. object recognition and object tracking).

Therefore, we model one MAR application as a combination of multiple tasks, which can be offloaded separately to powerful computing nodes nearby. Denote by M_k the number of tasks that the application from u_k contains. The computation complexity of each task is jointly determined by the input data size and the algorithm we adopt. For example, when recognizing object in images, different recognition algorithms and resolutions lead to different computational delays. We use c_k^i to denote the computational delay of the ith task from user u_k. Generally, an application profiler can be used to collect relevant metrics and estimate the computation amount of each task. Details about profiling are discussed in Section 8, here we focus on scheduling algorithms and assume that the profiler is given.

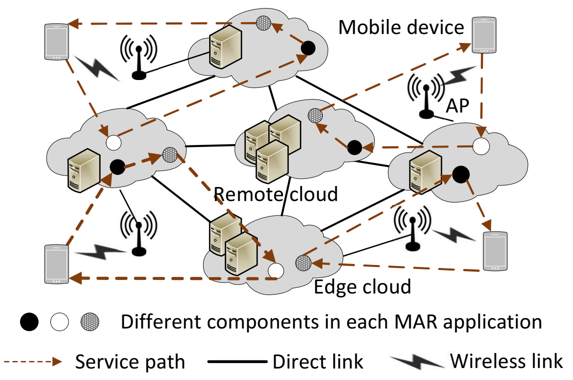


Fig. 2. An illustration of the cloud-edge network system.

3.3 Service delay model

The service delay of the kth user can be defined as

$$D_k = D_k^w + D_k^t + D_k^p, \tag{1}$$

where D_k^w is the wireless delay incurred by sending video frames from the kth user to its associated wireless access point; D_k^t is the core network delay caused by data communication between tasks; and D_k^p presents the total computational delay of all the tasks.

Wireless delay. The wireless delay consists of two parts: the upload delay (the red line in Fig. 3), which is caused by transmitting the input data from a user to its associated wireless access point (AP), and the download delay (the blue line in Fig. 3), which is caused by transmitting the output data from an AP to a user. Each part is determined by the data size and wireless data rate between a user and an AP. Since the size of the output data is usually small, the download delay can be ignored [23]. Let R_k be the average wireless data rate between user u_k and its associated AP, and w_k be the size of the data which should be uploaded for u_k. Then, the wireless delay experienced by the kth user can be modeled as

$$D_k^w = w_k / R_k. \tag{2}$$

Core network delay. Since the core network delay is mainly determined by the aggregated traffic loads and the geo-distance between servers. The impact of the data transmission of a single user on the core network delay is negligible [24]. Therefore, we do not consider such an

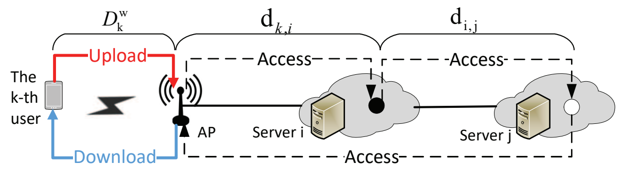


Fig. 3. An simple illustration of the service delay model.

impact in our core network delay model. To represent this delay, we first introduce a server assignment indicator $a_{k,i}^j$:

$$a_{k,i}^j = \begin{cases} 1 & \text{if the } i\text{th task from } u_k \text{ is placed at } s_j, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

For the simplicity of presentation, an AP is also regarded as a server with no computing ability in this paper. Let $d_{k,j}$ be the core network delay between server s_j and the AP that user u_k associates with.

The core network delay for user u_k consists of three parts (see Fig. 3 for example): 1) data transmission delay between the AP that u_k associates with and the edge server that the first task of u_k is assigned to, 2) communication delay between consecutive tasks, and 3) transmission delay of the final output data between the server that the last task of u_k is assigned to and the AP. More specifically, we have

$$\begin{aligned} D_k^t = &\sum_{j=1}^{N} a_{k,1}^j d_{k,j} + \sum_{m=2}^{M_k}\sum_{i=1}^{N}\sum_{j=1}^{N} a_{k,m-1}^i a_{k,m}^j d_{i,j} \\ &+ \sum_{j=1}^{N} a_{k,M_k}^j d_{k,j}. \end{aligned} \tag{4}$$

Computational delay. The computational delay is closely related to the computational complexity of a task and available computational resources on servers [23]. We use Q_j to denote the available computational resources on the jth server. Let $r_{k,i}^j$ be the percentage of server j's computational capacity being allocated to the ith task of user u_k. Then $Q_j \cdot r_{k,i}^j$ represents the amount of the computational resources allocated to the ith task of user u_k on server s_j. Therefore, the computational delay experienced by u_k can be modeled as:

$$D_k^p = \sum_{i=1}^{M_k}\sum_{j=1}^{N} a_{k,i}^j \frac{c_k^i}{Q_j r_{k,i}^j}. \tag{5}$$

3.4 Problem formulation

Combining Eqs. (1), (2), (4), and (5) together, we have

$$\begin{aligned} D_k = &\frac{w_k}{R_k} + \sum_{j=1}^{N} a_{k,1}^j d_{k,j} + \sum_{i=1}^{M_k}\sum_{j=1}^{N} a_{k,i}^j \frac{c_k^i}{Q_j r_{k,i}^j} \\ &+ \sum_{m=2}^{M_k}\sum_{i=1}^{N}\sum_{j=1}^{N} a_{k,m-1}^i a_{k,m}^j d_{i,j} + \sum_{j=1}^{N} a_{k,M_k}^j d_{k,j}, \end{aligned} \tag{6}$$

and the total service delay of all users is:

$$D = \sum\nolimits_{u_k \in \mathcal{U}} D_k. \tag{7}$$

On designing a scheduler, we aim to minimize the average service delay, which is equivalent to minimizing the total service delay. Denote $\mathcal{A} = \{a_{k,i}^j | u_k \in \mathcal{U}, s_j \in \mathcal{N}, i \in [1, M_k]\}$ as the set of server assignments. Denote $\mathcal{R} = \{r_{k,i}^j | u_k \in \mathcal{U}, s_j \in \mathcal{N}, i \in [1, M_k]\}$ as the set of resource allocations, the optimization problem is:

$$\begin{aligned} \mathcal{P}_1 : \min_{\{\mathcal{A},\mathcal{R}\}} \quad & D = \sum\nolimits_{u_k \in \mathcal{U}} D_k. \\ s.t. \quad & C_1 : \sum\nolimits_{j=1}^{N} a_{k,i}^j = 1, u_k \in \mathcal{U}, i \in [1, M_k] \\ & C_2 : \sum\nolimits_{k=1}^{M} r_{k,i}^j = 1, s_j \in \mathcal{N}, i \in [1, M_k] \\ & C_3 : a_{k,i}^j = \{0, 1\}, u_k \in \mathcal{U}, s_j \in \mathcal{N} \end{aligned} \tag{8}$$

.

The constraints C_1 and C_3 ensure that each task is assigned to one and only one server, and C_2 is the constraint on servers' capacity. By reducing the NP-complete Generalized Assignment Problem (GAP) [19] to M^{3}AS, we have the following theorem.

***Theorem** 1:* The M^{3}AS problem is NP-hard.

Proof. We prove this theorem by reduction from the Generalized Assignment Problem (GAP).

Definition 1 (Generalized Assignment Problem). Given a pair (E, S) where E is a set of n machines and S is a set of m tasks. Each machine $e_j \in E$ has capacity b_j, and we use $w_{k,j}$ and $p_{k,j}$ to denote the cost and the profit incurred by performing task s_k on machine e_j. The generalized assignment problem examines the maximum profit assignment of tasks to machines such that each task is assigned to precisely one machine subject to capacity restrictions on the machines.

Given an instance of GAP, we construct an instance of M^{3}AS as follows. We let $N = n$, $M = m$ and $Q_j = b_j$ in M^{3}AS. We assume that all offloaded applications contain only one task. For $u_k \in \mathcal{U}$, wireless delay D_k^w and computational delay D_k^p can be infinitely close to zero by setting uploaded data size and task computational complexity to a sufficiently small value. For the kth user, the core network delay between s_j and its associated AP ($d_{k,j}$) is set to be $P_{max} + 1 - p_{k,j}$, where $P_{max} = \max\{p_{k,j} : k \in [1, M], j \in [1, N]\}$. The core network delay between s_i and s_j can be any reasonable value. Let $r_{k,1}^j = \frac{w_{k,j}}{Q_j}$, where $w_{k,j}$ is the resource demand of task s_k' on machine e_j in GAP.

Combining these together, the objective of M^{3}AS is $\min \sum_{k=1}^{M}\sum_{j=1}^{N} a_{k,1}^j (p_{max} + 1 - p_{k,j})$, since P_{max} is fixed, the objective of M^{3}AS is then reduced to maximizing the assignment-profit under the constraint that the total resource demand of tasks assigned to s_j cannot exceed its capacity Q_j. Then, the problem is essentially GAP. It is not hard to see that the construction can be finished in polynomial time. Thus, we reduce solving the NP-complete GAP problem to solving a special case of M^{3}AS, implying that M^{3}AS is NP-hard. The theorem follows immediately.

4 Mutas for Offline M^{3}AS

4.1 Problem transformation and relaxation

Optimization problem $\mathcal{P}_1$ is a mixed integer nonlinear programming, and it is non-convex due to its non-convex quadratic terms in Eq. (4). However, we note that for a certain user u_k, the communication delay between its $(m-1)$th and mth tasks can be modified as follows:

$$\begin{aligned}&\sum_{i=1}^{N}\sum_{j=1}^{N} a_{k,m-1}^{i} a_{k,m}^{j} d_{i,j}\\ =&\sum_{i=1}^{N}\sum_{j=1}^{N} d_{i,j}\max[(a_{k,m-1}^{i}+a_{k,m}^{j}-1),0],\end{aligned} \tag{9}$$

where the equality holds because $a_{k,i}^{j}$ is binary. This converts the non-convex term to a convex form.

Therefore, we perform the following two-step transformation and relaxation on problem $\mathcal{P}_1$:

- Replace the communication terms in $\mathcal{P}_1$ with Eq. (9).
- Relax binary variables $a_{k,i}^{j}$'s to continuous variables $\widetilde{a}_{k,i}^{j}$'s. Let $\widetilde{\mathcal{A}}$ denote $\{\widetilde{a}_{k,i}^{j} | u_k \in \mathcal{U}, s_j \in \mathcal{N}, i \in [1, M_k]\}$.

The relaxed form of D_k in Eq. (6) is:

$$\begin{aligned}\widetilde{D}_k =& \frac{w_k}{R_k} + \sum_{j=1}^{N}\widetilde{a}_{k,1}^{j} d_{k,j} + \sum_{i=1}^{M_k}\sum_{j=1}^{N}\widetilde{a}_{k,i}^{j}\frac{c_k^i}{Q_j r_{k,i}^{j}}\\ &+\sum_{m=2}^{M_k}\sum_{i=1}^{N}\sum_{j=1}^{N} d_{i,j}\max[(\widetilde{a}_{k,m-1}^{i}+\widetilde{a}_{k,m}^{j}-1),0]\\ &+\sum_{j=1}^{N}\widetilde{a}_{k,M_k}^{j} d_{k,j}.\end{aligned} \tag{10}$$

The relaxed $\mathcal{P}_1$ can be presented as follows:

$$\begin{aligned}\mathcal{P}_2 : \quad &\min_{\{\widetilde{\mathcal{A}},\mathcal{R}\}} \quad \widetilde{D} = \sum_{u_k\in\mathcal{U}} \widetilde{D}_k\\ &s.t. \quad C_1,\ C_2\\ &\qquad \widetilde{C}_3 : 0 \le \widetilde{a}_{k,i}^{j} \le 1; u_k \in \mathcal{U}, s_j \in \mathcal{N}.\end{aligned} \tag{11}$$

4.2 Formal analysis

In this section, we first introduce a central element in convex optimization: the notion of convex function.

***Theorem** 2:* Suppose a function $f : R^n \to R$ is twice differentiable, then f is convex if and only if its domain is a convex set and its Hessian matrix is positive semidefinite: i.e., for any $x \in D(f)$, $\Delta^2 f(x) \succeq 0$.

Proof: The detailed proof can be found in [25]. □

Similar to previous work [3], we then partition the variables in $\mathcal{P}_2$ into two parts of variables, and prove that the relaxed problem $\mathcal{P}_2$ is convex with respect to these two parts, respectively. Based on the previous theorem, we have the following two lemmas:

Algorithm 1: The Mutas Algorithm

```
Input: convergence condition $\tau$; initial assignment
       $\widetilde{\mathcal{A}}_0$ ; maximum iteration number $G_k$
Output: server assignment $\mathcal{A}$, resource allocation $\mathcal{R}$
1  $\widetilde{\mathcal{A}} \leftarrow \widetilde{\mathcal{A}}_0$, $i \leftarrow 0$;
2  while True do
3      $\mathcal{R} \leftarrow$ update $\mathcal{R}$ with fixed $\widetilde{\mathcal{A}}$;
4      $\widetilde{\mathcal{A}} \leftarrow$ update $\widetilde{\mathcal{A}}$ with fixed $\mathcal{R}$;
5      $\widetilde{D}(i) \leftarrow$ compute the total delay with $\widetilde{\mathcal{A}}$ and $\mathcal{R}$;
6      If $|\widetilde{D}(i) - \widetilde{D}(i-1)| \le \tau$ or $i \ge G_k$ then break;
7      $i \leftarrow i+1$;
8  recover a binary solution $\mathcal{A}$ according to Eq. (13) ;
9  $\mathcal{R} \leftarrow$ update $\mathcal{R}$ with fixed $\mathcal{A}$;
10 return $\mathcal{A}, \mathcal{R}$
```

***Lemma** 1:* The problem $\mathcal{P}_2$ is strictly convex with respect to the relaxed server assignment $\widetilde{\mathcal{A}}$.

Proof: For any given $\widetilde{a}_{k,i}^{j}$, the objective function only contains linear terms and the max function of $\widetilde{a}_{k,i}^{j}$. The max function is convex, and the addition of convex functions remains convex. The constraint C_1 is linear. The constraint C_2 is irrelevant to $\widetilde{\mathcal{A}}$. Therefore, $\widetilde{D}$ is strictly convex with respect to $\widetilde{\mathcal{A}}$. □

***Lemma** 2:* The problem $\mathcal{P}_2$ is strictly convex with respect to the resource allocation $\mathcal{R}$.

Proof: For any feasible variables $r_{k,i}^{p}, r_{v,j}^{q}$, $\forall u_k, u_v \in \mathcal{U}$, $\forall s_p, s_q \in \mathcal{N}$,

$$\frac{\partial^2 \widetilde{D}}{\partial r_{k,i}^{p}\partial r_{v,j}^{q}} = \begin{cases}\frac{2c_k^i Q_p a_{k,i}^{p} Q_q}{(Q_p r_{k,i}^{p})^3} & \text{if } k=v, i=j, p=q,\\ 0 & \text{otherwise.}\end{cases} \tag{12}$$

The Hessian matrix $H = [\frac{\partial^2 \widetilde{D}}{\partial r_{k,i}^{p}\partial r_{v,j}^{q}}]$ is symmetric and positive definite. According to Theorem 2, function f is strictly convex if its Hessian matrix H is positive definite for all the variables. The constraint C_2 is linear. The constraints C_1, $\widetilde{C}_3$ are irrelevant to $\mathcal{R}$. Therefore, $\widetilde{D}$ is strictly convex with respect to $\mathcal{R}$. □

Lemma 1 confirms that, when server assignment $\widetilde{\mathcal{A}}$ is fixed, the total service delay $\widetilde{D}$ can be decreased by optimizing resource allocation $\mathcal{R}$. Similarly, with Lemma 2, when resource allocation $\mathcal{R}$ is fixed, the processing delay of each task on a server can be determined, thus the relaxed service delay $\widetilde{D}$ can be reduced by optimizing server assignment $\widetilde{\mathcal{A}}$. Based on these intuitions, we developed a Multi-task application scheduling (Mutas) algorithm, with further details in the next subsection.

4.3 Mutas design

The main idea behind Mutas is fixing one variable while optimizing the other one, after several iterations, the objective function converges to a certain value. For example, we will derive a new server assignment after optimizing $\widetilde{\mathcal{A}}$ by the block coordinate descent method

Algorithm 2: The Mutas+SA Algorithm

```
Input: cooling parameter $\alpha$; initial temperature
       $T_{max}$; terminating temperature $T_{min}$
Output: server assignment $\mathcal{A}$, resource allocation $\mathcal{R}$
1  randomly generate an initial assignment $\widetilde{\mathcal{A}}_{old}$;
2  $\mathcal{A}_{old}, \mathcal{R}_{old} \leftarrow$ Mutas($\widetilde{\mathcal{A}}_{old}$);
3  $f \leftarrow D(\mathcal{A}_{old}, \mathcal{R}_{old})$; $T \leftarrow T_{max}$;
4  while $T > T_{min}$ do
5      randomly generate a new assignment $\widetilde{\mathcal{A}}_{new}$ in
         the neighborhood of $\widetilde{\mathcal{A}}_{old}$;
6      $\mathcal{A}_{new}, \mathcal{R}_{new} \leftarrow$ Mutas($\widetilde{\mathcal{A}}_{new}$);
7      $f' \leftarrow D(\mathcal{A}_{new}, \mathcal{R}_{new})$; $\Delta d = f - f'$;
8      if $\Delta d \leq 0$ then
9          $\mathcal{A}_{old} \leftarrow \mathcal{A}_{new}$; $f \leftarrow f'$;
10     else if $Rand(0,1) < exp(-\frac{\Delta d}{T})$ then
11         $\mathcal{A}_{old} \leftarrow \mathcal{A}_{new}$; $f \leftarrow f'$;
12     $T \leftarrow \alpha \cdot T$;
13 $\mathcal{A} \leftarrow \mathcal{A}_{old}$; $\mathcal{R} \leftarrow \mathcal{R}_{old}$;
14 return $\mathcal{A}, \mathcal{R}$
```

with fixed $\mathcal{R}$, then this new assignment can be leveraged to generate an optimized $\mathcal{R}$ conversely. The above iterative processes will continue until a pre-defined maximum number of iterations G_k has been reached or no significant improvement can be achieved. The pseudo code of Mutas is shown in Algorithm 1. After solving Problem $\mathcal{P}_2$, $\widetilde{a}_{k,i}^j$ are converted to $a_{k,i}^j$ according to

$$a_{k,i}^j = \begin{cases} 1 & \text{if } j = \arg\max_{s_j \in \mathcal{N}} \widetilde{a}_{k,i}^j, \\ 0 & \text{otherwise.} \end{cases} \tag{13}$$

Lemmas 1 and 2 lay the foundation for the convergence of the Mutas algorithm. Since $\widetilde{D}$ is convex with respect to each block of variables, the convergence of the proposed algorithm is guaranteed according to [26].

The last step is binary recovery which provides a feasible solution to the original optimization problem $\mathcal{P}_1$. The recovered solution is no longer optimal and can be further improved by simulated annealing (SA).

Complexity analysis. The Mutas algorithm converges to the optimal value for the transformed problem $\mathcal{P}_2$ in a linear rate, in other words, the sequence of the objective function values converges to the optimum in an $O(\frac{1}{\epsilon})$ rate, where ϵ is the parameter to control the accuracy. The Mutas algorithm is developed based on the alternating minimization method, and the convergence and the optimality were proved in [27]. The corresponding evaluations on the time cost of the Mutas algorithm is given in Section 6.2.

4.4 Augmenting Mutas with simulated annealing

As we mentioned before, the last step of Mutas is to recover a set of binary variables. However, the recovered solution sometimes may be not so satisfactory. To improve Mutas, we leverage simulated annealing (SA) which can help Mutas to escape from local minimums. Note that there are many other alternative algorithms such as genetic algorithm (GA) and particle swarm optimization (PSO). According to our previous research [28], PSO has advantage over GA in both convergence speed and accuracy. However, when compared with SA, PSO has more parameters (inertia weight, learning factor and so on) due to its complex design, making the setting of parameters a very technical thing. Taking all the considerations into account, we adopt SA for its easy implementation and fast convergence.

The Mutas+SA algorithm is shown in Algorithm 2. Note that, α is the cooling parameter in SA, T_{max} and T_{min} are initial and terminating temperatures, respectively. At the beginning, the simulated temperature T equals to T_{max} then decreases in each iteration, when $T = T_{min}$ SA iterations terminate. Consequently, the SA process has no bounded time complexity. Instead, its time complexity depends on the parameter settings.

Generally, a longer searching time leads to a better solution, but also increases the computational overhead of the scheduling algorithm. We can flexibly control the tradeoff between accuracy and running time by adjusting the algorithm parameters. In Section 6, we will investigate the impact of the SA cooling parameter on the performance and time cost of the offline algorithm.

Mutas and SA each serves an important role in the overall algorithm design. First, Mutas provides a suitable starting point for SA. If we start from some randomly chosen point in the solution space, SA can converge to some sub-optimal solution. Second, an appropriate starting point helps reduce the number of SA iterations. Further numerical evaluation of the contribution of the SA step is given in Section 6.2.

5 OnMutas for Online M^{3}AS

The Mutas algorithm calculates the whole service path for a batch of users, and thus servers need to reserve resources for tasks assigned to it. It is a waste of resources since the computing capacity reserved for a task could be utilized by other tasks before this task arrives. Based on this intuition, in this section, we discuss the online version of the M^{3}AS problem, where user applications arrive over the time, and we present the Online Multi-task application scheduling (OnMutas) algorithm that makes scheduling decisions on the task level.

5.1 Intuition

In the online case, we partition time into discrete time slots of equal length. At the beginning of each time slot, a scheduling decision should be made for tasks that are ready to be executed. As shown in the Fig. 4, two tasks (marked with stars in the figure) from users u_3 and u_4, respectively, can be scheduled at the beginning of the current time slot. User u_2 still has task being executed at

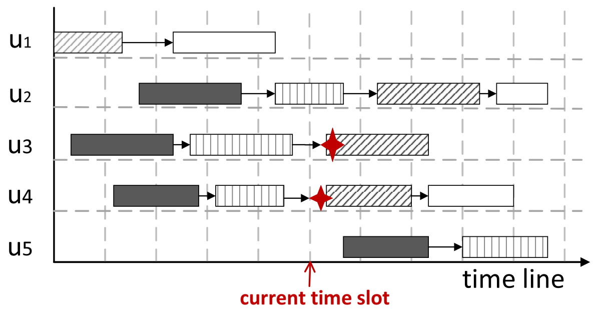


Fig. 4. An illustration of the online scenario where each rectangle represents a task and each arrow represents dependency. OnMutas makes scheduling decisions on the task level.

this moment, hence the succeeding tasks are not ready for execution due to data dependency.

Lemma 2 can beextended to the following theorem:

***Theorem** 3:* When each $a_{k,i}^j$ is fixed, the corresponding optimal resource allocation to $\mathcal{P}_1$ is

$$r_{k,i}^{j\ *} = \frac{\sqrt{a_{k,i}^j c_k^i}}{\sum_{u_k \in \mathcal{U}} \sum_{i=1}^{M_k} \sqrt{a_{k,i}^j c_k^i}}. \tag{14}$$

Proof: The convexity of $\mathcal{P}_1$ over $\mathcal{R}$ is proved by Lemma 2. The corresponding Lagrangian of $\mathcal{P}_1$ is:

$$L(\mathcal{R}, \mu) = D - \sum_{j=1}^{N} \mu_j (\sum_{k=1}^{M} \sum_{i=1}^{M_k} r_{k,i}^j - 1). \tag{15}$$

By solving the KKT condition [29] of Eq. (15), we can obtain the optimal solution $r_{k,i}^{j\ *} (i = [1, M_k], u_k \in \mathcal{U}, s_j \in \mathcal{N})$ and the corresponding optimal value D_{min}. □

Theorem 1 shows that, when the placement for each task is fixed, the corresponding resource allocation problem for these tasks is deterministic.

5.2 Branch and bond-based onMuta

Note that any application cannot have more than one task that is ready for execution at the beginning of any time slot. We assume there are m tasks to be scheduled, and the computational complexities of them are c_1, c_2, ..., and c_m, respectively. If no confusion is caused, we also use $pre(c_i)$ to denote the server where the predecessor of task c_i is placed.

Without loss of generality, we assume that the workloads $c_1, c_2, ..., c_m$ are sequentially determined to be placed at the start time of a time slot. Therefore, we only need to solve the following subproblem: given the service assignment for c_1, c_2, ..., and c_k, which server should c_{k+1} be placed on? Based on this subproblem definition, we have the following theorem:

***Theorem** 4:* Let d_k indicate the total amount of delays for the k tasks having been placed, which can be easily calculated by solving the corresponding convex optimization problem based on Theorem 1. The lower

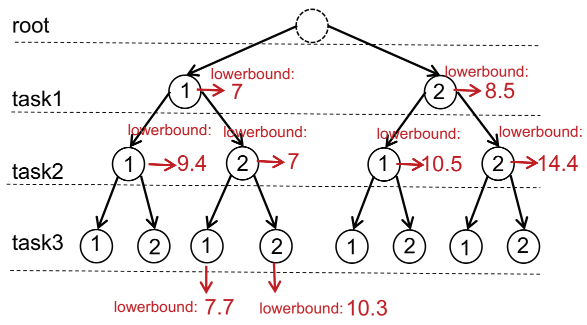


Fig. 5. An example branch and bound-based OnMutas.

bound on the total amount of delays for the remaining $(m - k)$ tasks can be presented as:

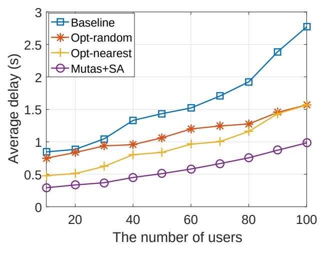
$$u_k = d_k + \sum_{i=k+1}^{m} [\min_{s_j \in \mathcal{N}} \{\frac{c_i}{Q_j} + d_{pre(c_i),j}\}]. \tag{16}$$

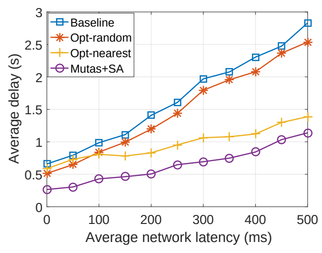
Proof: In Eq. (16) all the remaining $(m - k)$ tasks are provisioned with 100% capacity of a server, and thus u_k provides a lower bound on the total amount of delays for the remaining $(m - k)$ tasks. □

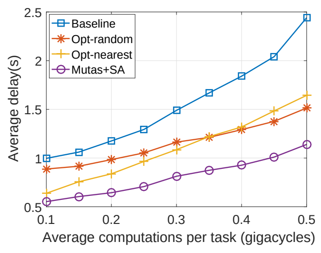
Based on the above problem transformation, we then develop OnMutas to determine the optimal values of $\{a_{k,i}^j\}$. Our algorithm is based on the branch and bound approach. Notice that the solutions of subproblems usually constitute a search tree for the original problem. When applying depth-first search over subproblems, we adopt a greedy heuristic by searching the one with the smallest lower bound first. Each time when a leaf node in the search tree is reached, all other branches with larger values of lower bounds are pruned. The optimal solution will be found until all the leaf nodes have been either searched or pruned.

Fig. 5 shows an example of the proposed algorithm. Assuming that there are 3 tasks c_1, c_2, and c_3 with computational complexities equal to 3, 2, and 1, respectively; there are 2 servers s_1 and s_2 with capacities equal to 2 and 1, respectively; the data transmission delays[1] are $d_{pre(c_1),1} = 1$, $d_{pre(c_1),2} = 2$, $d_{pre(c_2),1} = 2$, $d_{pre(c_2),2} = 1$, $d_{pre(c_3),1} = 1$, and $d_{pre(c_3),2} = 1$. As shown in the figure, we are going to determine which server should c_1 be placed on? If task c_1 is placed on s_1, the lower bound of this node is $\frac{3}{2} + 1 + \frac{2}{1} + 1 + \frac{1}{2} + 1 = 7$; similarly, the lower bound of placing c_1 on s_2 is 8.5. We will search the node with the smallest lower bound first. The first leaf node we visit has a lower bound of 7.7, under the condition that c_1 and c_3 are placed on s_1 while c_2 is placed on s_2. After that all the branches with lower bounds larger than 7.7 will be pruned. For all the leaf nodes in the figure, six of them are pruned (gray nodes in the figure). Among the rest two leaf nodes, the one with the smallest lower bound indicates the optimal solution.

1. If c_i has no predecessor, the delays indicate the data transmission delay from the AP to the servers.


(a) User Number


(b) Average Network Latency


(c) Computation Amounts

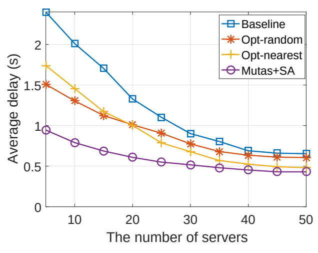

(d) Server Number

Fig. 6. Impact of different parameters on the performance of the offline algorithm

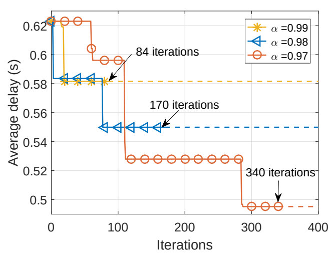


Fig. 7. The effect of the cooling parameter α

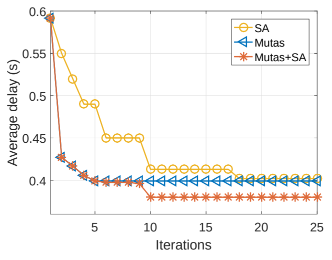


Fig. 8. The contribution of the Mutas+SA

6 Simulation Results

In this section, we evaluate the performance of the proposed offline and online algorithms by extensive simulations. The simulation is conducted on a Dell OptiPlex 7050 desktop computer with an Intel i5- 7500@3.4GHz CPU and 8GB RAM.

6.1 Simulation setup

We simulate an edge network with 50 wireless access points. The wireless data rates of users are uniformly distributed between 1 to 10 Mbps. According to [3], the computing capacities of edge servers are uniformly generated between 2 GHZ and 5 GHZ, while the computing capacities of edge servers are uniformly generated between 8 GHZ and 12 GHZ. For each task being received by a server, we specify its amount of computation needed as the number of CPU cycles. Particularly, if a task with 1 gigacycles is offloaded to a server provisioned with 2GHz capacity, its computation delay will be 0.5 seconds.

In the offline case, there are 20 servers, both edge servers and cloud servers are included. The network latencies between wireless APs and edge servers are generated according to a normal distribution with the mean of 50ms. The network latencies between the wireless access points and the cloud servers are also generated based on a normal distribution but with the mean of 150ms. The edge servers and cloud servers are randomly deployed in the network. In the online case, if not specified explicitly, the task queue length is set to be 10.

Since the problem formulation is unique, other state-of-the-art algorithms can be hardly found. As for dispatching policies, Nearest is the most friendly approach to distributed systems. For example, task assignment policy in [24] is to greedily dispatch a job to the server which brings the least increase to the total response time. As the mobile device only needs to communicate with the nearest edge-clouds, most of the works (e.g., [30], [31]) on edge clouds adopted Nearest as the job dispatching policy. The centralized scheduling algorithms aim to find the optimal task assignment solution. For example, the authors in [3] designed an edge network orchestration algorithm named FACT which optimizes the edge server assignment and video frame resolution selection for MAR users. The problem is convex and the optimal solution can be derived by convex optimization. Random is close to the practical scenarios where the global information about network topology is unknown.

As for the resource allocation policy, one popular resource allocation policy is equal division (the available computational resources on a server are evenly shared by the tasks assigned to the server), which is commonly used by the approaches designed for parallel multi-user systems (e.g., [32], [3]). To ensure more efficient utilization of computing resources, the work in [33] proposed a workload placement algorithm that decides which edge cloud servers the mobile programs are placed on and how much computational capacity is provisioned to execute each program. The workload placement algorithm is developed based on brute-force search method and the optimal scheme can be found using convex optimization.

Therefore, we design the comparisons as the combinations of these typical dispatching and resource allocating policies. In our simulations, we compare our algorithms with four heuristics (Baseline, Opt-random, Nearest, and Opt-nearest) summarized in Table 3. Each algorithm contains two strategies for server assignment and resource management, respectively. Table 3 shows which strategy is used in each algorithm.

In the table, by "greedy server assignment", we mean, for the first task in an application, it will be sent to the server with the smallest data upload delay; for other task, the server we choose will be the nearest one from where its predecessor is located. By "optimal resource allocation", we mean, if the server assignment of all tasks is fixed, we can further optimize the total service delay by solving the resource allocation problem.

In Mutas+SA, the initial temperature is generally set to be a value that results in average acceptance probability equal to 0.8 for bad mutations from initial solution [34].

TABLE 1
Algorithms in Comparison (opt is short for optimal)

	resource allocation		server assignment		
	average	opt.	random	greedy	opt
Mutas/OnMutas		✓			✓
Baseline	✓		✓		
Opt-random		✓	✓		
Nearest	✓			✓	
Opt-nearest		✓		✓	

Hence, we randomly generate lots of cost-decreasing mutations and compute the average decrease in cost $\Delta cost$, then the initial temperature T is set to be $-\frac{\Delta cost}{ln(0.8)}$. When evaluating the performance of proposed offline algorithm, the cooling parameter α is set to 0.97, and the initial temperature and terminating temperature are set to 10 and 1, respectively.

6.2 Offline performance

To evaluate the impact of cooling parameter on the performance of the offline algorithm, we adjust the value of cooling parameter and solve the scheduling problem by SA based on the solution derived by Mutas. The experiment results are shown in Fig. 7. Though a larger value of α usually generates a better solution, but it can also slow the speed of SA convergence, incurring larger time cost of the algorithm itself. We can reduce the number of iterations at a slight sacrifice of accuracy.

Despite the fact that the derived solution is not optimal, we observe that the distance from the optimum are generally negligible in our experiments. For example, in the case depicted by Fig. 9, two edge servers and a cloud center are deployed, there are three users connected to the nearby APs, each with a two-transform application to offload. The parameter settings including server capacity, task computation complexity, and network latency are given in Fig. 9, under which, the average delayS generated by Mutas+SA and the brute force method are 0.85s and 0.82s respectively, we believe that this time difference of 0.03s does little harm to the user perception.

To demonstrate the role and contribution of Mutas and SA in the offline algorithm, we first run them separately to provide a feasible solution to the original problem. The convergence process of these algorithms is shown in Fig. 8. First, Mutas algorithm converges to the local optimum quickly in several iterations, Second, with an appropriate starting point, SA pushes the solution to one that is closer to the optimal solution. We can observe that when combined together in the proposed manner, Mutas and SA each serves an important role in the overall algorithm design. For further performance evaluation, we also consider the following factors:

The impact of the number of users. We first evaluate the performance of Mutas with different numbers of users. The MAR application of each user is independent from all others. Fig. 6(a) reflects the impact of the number of users on the performance of four different algorithms. When the number of users increases, servers experience more workloads. Thus, the service delay increases. When

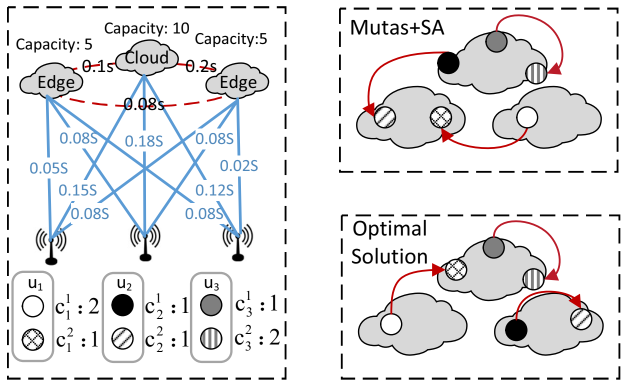


Fig. 9. The solution from Mutas+SA vs the Optimum.

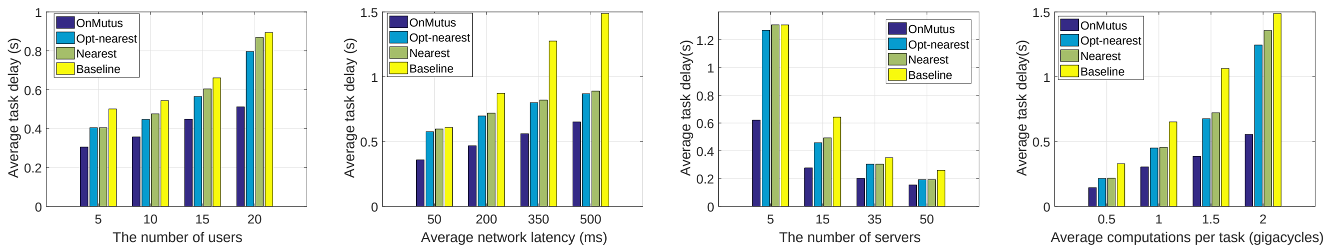
compared with the other algorithms, the Mutas algorithm achieves the smallest delay. As compared with the Baseline algorithm, the Mutas algorithm gains up to 65% service delay reduction when the number of users is 100. Since Baseline and Opt-random both adopt random server assignment, the difference between them reflects the improvement gained from the optimization of resource allocation. We can observe that when workload increases, such improvement becomes more obvious.

Meanwhile, the gap between Opt-nearest and Opt-random narrows as task number increases. With the amount of workloads increases, the resource contention on each server leads to a longer processing time. As the ratio of network latency to the total delay decreases, the greedy server assignment gradually loses its advantage over the random server assignment.

The impact of network latency. Fig. 6(b) shows the impact of the average network latency on the performance of different algorithms. Here, the average network latency reflects the average transmission delay between servers. As shown in the figure, the average service delay increases when the average network latency increases. Mutas significantly reduces the delay by 61% compared to the Baseline. The difference between Opt-random and Baseline reflects the improvement from the optimization of resource allocation. Note that, the variance of the network latency has no impact on the gap between Opt-random and Baseline, since network latency is irrelevant to resource contention on servers.

Besides, the improvement from greedy task dispatching becomes significant as the network latency increases. Since service delay comprises of data transmission delay and computational delay; as network latency increases, data transmission becomes the dominant component, hence the improvement due to the optimization of data transmission delay becomes more obvious.

The impact of the amounts of computations. The performance of four algorithms with varying amounts of computations is shown in Fig. 6(c). When more workloads are included in each task, the service delay increases. Mutas can reduce the average service delay by 48%, compared to that of Baseline. Compared to Opt-nearest, the delay reduction achieved by Mutas is about 22%. When comparing baseline with Opt-random, we



(a) User Number (b) Average Network Latency (c) Server Number (d) Computation Amounts

Fig. 10. Impact of different parameters on the performance of the online algorithm in terms of average task delay

can conclude that, with larger computation amounts, the delay reduction brought by resource allocation optimization becomes more obvious. Note that, the greedy server assignment can even perform worse than the random dispatching when servers experience more workloads.

The impact of the number of servers. The impact of the number of servers is shown in Fig. 6(d). Obviously, with more servers, all algorithms have smaller delays. Compared with other algorithms, Mutas achieves the smallest service delay. However, the performance gain becomes very small when adding more servers after the server number reaches 30. We can see that the performance gain from the resource allocation optimization is greater when the number of servers is limited.

TABLE 2
Mutas Runtime Evaluation

	Mutas(Sec)		SA(Sec)	
Task num	Avg runtime	Iterations	Avg runtime	Iterations
10	0.142	4	0.009	84
20	0.271	5	0.017	84
30	0.396	7	0.024	84
40	0.535	9	0.032	84
50	0.813	10	0.048	84

The running time of the offline algorithm. The running time of other algorithms in comparison is negligible in most cases, so we only record the average running time of the proposed algorithm in the table 2, along with the corresponding iteration number of Mutas and SA. For Mutas, when task number rises from 10 to 50, it always converges within 10 iterations, however, since each iteration involves solving a convex problem, the running time increases significantly with problem scale. For SA, with more tasks, the required iterations is fixed since it only depends on the parameter settings, the running time also shows a small increase. It can be observed from table 2 that, even requires more iterations, SA has a fast running speed than Mutas. At the meantime, as shown in Fig. 8, the first few iterations brings the majority of the decrease in objective function for Mutas. Based on these two observations, when task number grows, we suggest optimizing the proposed algorithm by cutting the iterations of Mutas to save running time, which means we have to accept a temporarily worse starting point for SA, but as a compensation, we can add SA iterations by adopting a larger α.

6.3 Online performance in terms of average task delay

OnMutas is designed to schedule MAR tasks that are ready for execution at the beginning of the current time slot. In this subsection, we evaluate OnMutas in terms of average task delay. The average task delay includes two parts: the data transmission delay between its preceding task and itself, and the computational delay of the task.

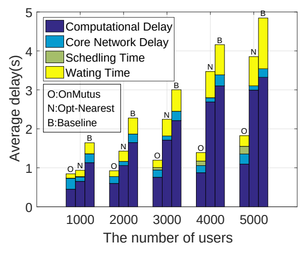
The impact of the number of users. Fig. 10(a) shows the impact of the user number on the performance of different algorithms. When the number of users in queue increases, servers experience more workloads. Thus, the service delay increases. Compared with other algorithms, OnMutas achieves the smallest delay. Compared with Baseline, OnMutas gains up to 47% service delay reduction when the number of users is 20. The performance gain of Opt-nearest over Nearest reflects the influence of the resource allocation optimization.

The impact of network latency. Fig. 10(b) shows the impact of the network latency on the performance of different algorithms. As shown in the figure, the average service delay increases when the average network latency increases. We can see that the network latency can greatly affect the performance of Nearest. For example, the performance of Nearest and Baseline are similar in the 50ms case, while the delay reduction of Nearest over Baseline is up to 41% in the 500ms case.

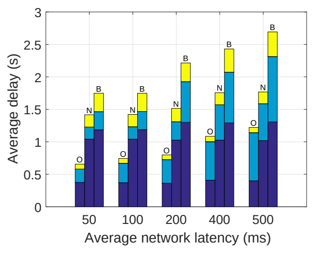
The impact of the number of servers. Fig. 10(c) shows the average service delay with different numbers of servers. When compared with the other algorithms, OnMutas achieves the smallest delay. Similar to the offline case, the performance gain of adding more servers is small after the server number reaches 15.

The impact of the amount of computations. Fig. 10(d) indicates that average delay is affected by the amount of computations. Note that a larger amount of computations leads to more delay reduction. For example, when the average amount of computations increases to 2 gigacycles, the delay reduction could be up to 60%.

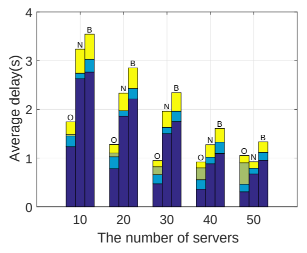
The running time of onMutas. Table 3 shows the running time of onMutas and depth-first search (DFS). We configure two edge servers and a remote cloud with task number rising from 10 to 30. When there are m computing tasks and n servers, the time complexity to exhaustively search the solution space is $O(n^m)$. Obviously, with more tasks, the running time of depth-first



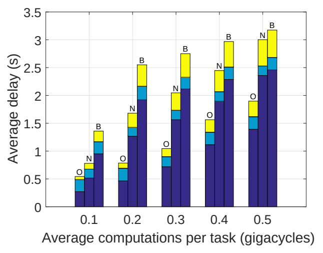
(a) User Number



(b) Average Network Latency



(c) Server Number



(d) Computation Amounts

Fig. 11. Impact of different parameters on the performance of the online algorithm in terms of average service delay

search increases significantly. The performance gain of OnMutas can be regarded as the advantage of adopting the branch and bound approach over DFS. The decreased complexity varies in different cases and can be hardly expressed in a unified mathematical form. According to our simulations, the average time cost reduction of OnMutas over DFS is up to 80%.

TABLE 3
OnMutas Runtime Evaluation

Avg runtime \ Algorithm / Task number	OnMutas(Sec)	DFS(Sec)
10	0.021 ± 0.014	0.093 ± 0.027
15	0.045 ± 0.029	0.159 ± 0.149
20	0.183 ± 0.110	0.674 ± 0.520
25	0.244 ± 0.226	1.452 ± 1.139
30	0.577 ± 0.359	1.970 ± 1.447

6.4 Online performance in terms of average service delay

As we mentioned before, the service delay includes wireless delay, core network delay, and computational delay. Since the wireless delay is irrelevant to the specific scheduling algorithm, we will not consider it in this subsection. We are also interested two other delays: scheduling time and waiting time. The former is the running time of OnMutas, while the latter is the waiting time when a task is in the task queue.

We first investigate the influence of the number of users on the performance of OnMutas. The results are shown in Fig. 11(a). When the number of user increases, servers experience more workloads, and thus both the processing time and the waiting time of each application increase. When the number of user increases, the length of task queue also increases, leading to a longer running time of OnMutas. Note that, the running time of other heuristics is negligible in our experiments.

In Fig. 11(b), when the average network latency increases, users experience longer service delays. Results show that network latency has no impact on the processing time and the waiting time of all tasks. As network latency increases, the proportion of data transmission delay increases, the performance of Opt-nearest hence gets better due to its optimization on server assignment.

In Fig. 11(c), when adding more servers into the system, the processing time and waiting time of tasks decrease. Meanwhile, the running time of OnMutas increases since adding servers enlarges the solution space.

Fig. 11(d) shows the average delay with different amounts of computations. Obviously, as more computations are included in each task, a longer processing time is needed. Note that, the delay reduction of OnMutas over other heuristics becomes more obvious as the amount of computations increases.

As a brief summary, we evaluated the performance of Mutas and OnMutas with extensive simulations. The results show that both our offline and online algorithms can greatly reduce the service delay of MAR systems.

7 System Design and Experiments

We present the architecture of our MAR platform in this section. As illustrated in Fig. 12, the requests are first submitted to the centralized network controller, which is deployed at the proximity of users. The monitoring service and profiling service running in the network controller keep collecting necessary information for the scheduling algorithm. Note that the task is the unit of the application for our system to make scheduling and optimization decisions. The tasks are then sequentially executed on the scheduled server and the analysis output will be sent back to the corresponding MAR users.

Monitoring Service. The system status monitor continuously collects system run-time information including network latency and server workload. The monitoring of users wireless data rates is also necessary since wireless bandwidth exhibits substantial dynamics over time. These information is then utilized by the scheduling algorithm to co-optimize the task offloading decision and resource allocation.

Profiling Service. The application profiler models the impact of input datasize on the task computation complexity. In prior studies, the computation complexity with respect to the resolution is formulated as a concave function for CNN-based object detection algorithms [3], [35]. It is reasonable because for a specific CNN, the input image size is fixed. A small CNN generally processes images with low resolution, contrarily, a large CNN takes images with high resolution as the input. Since the running time of CNN increases with the number of network layers, the running time of the deep-learning algorithm is indirectly determined by the image resolution, in other words, when the input data size is fixed, the computation complexity of image processing task

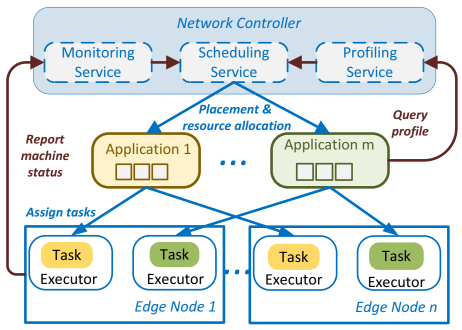


Fig. 12. The system architecture of our MAR platform.

is approximately the same. For other image-processing algorithms involved in MAR application (e.g. object tracking [36] and image rendering [37]), the algorithm complexity is also closely relevant to the amount of pixels in the image [38]. In conclusion, the running time of image processing is determined by the input data size. This observation allows us to build a profiling tool that learns an accurate relationship between input datasize and computation complexity (c_k^i in Section 3.2) for each task in AR applications. Since different applications tend to have different profiles, we add a profiling phase to the deployment of every new type of AR application. For each AR request, the profiling service first estimates the corresponding computation complexity of all tasks involved according to its input datasize, the scheduling module then relies on the information for task dispatching and resource management.

Scheduling Service. The third module is the scheduler, which is responsible for the task dispatching and resource management. The key components of the scheduler are Mutas and onMutas algorithms. When the Mutas algorithm solves the scheduling problem, prerequisite information about the computation complexities of all the tasks involved is required. The offline centralized scheduler applies to small MAR systems since the scheduling delay increases with the network scale, while the online scheduler is more general. The onMutas algorithm is invoked periodically or based on the task queue length, both of which are adjustable.

7.1 Evaluation setup

Our experiment consists of five virtual machines, among which four are edge/cloud computing nodes, and the other one is the network controller. The five virtual machines are connected through virtual routers which are emulated by virtual machines.

In our experiments, the link latency is emulated by linux TC (traffic control) [39]. We roughly separate MAR applications into four tasks, preprocessing (including resizing, decoding and denoising), object extracting, object recognizing and image construction. Fig. 13 depicts two MAR applications we support, in which one AR chain include complete four stages while the other do not require the extracting of certain objects. Our implementation uses OpenCV [40] for object abstraction,

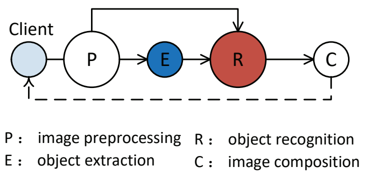


Fig. 13. Two MAR applications we support in our experiment: P-E-R-C and P-R-C.

YOLO [41], a pre-trained neural network, for object recognition and OPENGL for image construction. We first use an offline process to build a default profile on a server with 32-core CPU and 64G memory. The profiler executes instrumented tasks multiple times with different inputs and measures metrics including execution time, input/output data size, etc. More specifically, in order to make CNN support different input datasizes, we load five different CNN models with various input resolutions. Fig. 14 shows the impact of the input data size on the task computational delay. We observe that for most image-related operations, the relationship between input datasize and the computation complexity can be modeled as concave functions.

7.2 Evaluation results

Fig. 15 shows the impact of the user number on the average service latency. We compare onMutas with two heuristics (Opt-random, Opt-nearest) mentioned before. When the number of user rises from 2 to 4, the response time shows slight variance, however, the average delay significantly increases with heavier system load. We found that the performance of Opt-nearest highly relies on the user distribution, Opt-nearest can even perform worse than Opt-random with unbalanced workload. Similar to our simulations, the performance gain of OnMutas over other algorithms is obvious. We also optimize the Mutas algorithm by allocating reserved resources not in use to running tasks on the server. The optimized Mutas algorithm performs slightly better than onMutas in a few cases (e.g. when the number of users equals to 8). Our experiment result shows that the performance of these two algorithms is overall similar when all the requests arrive at about the same time.

8 Discussions

Handling DAG jobs. We notice that the interdependency of tasks in an application exhibits more complicated patterns. For example, in [42], the car plate detection application is partitioned into multiple tasks, whose dependency is modeled as a directed acyclic graph (DAG). Here we take a step further to extend onMutas for DAG-based task graphs. For a given DAG $G = < V, E >$, where V is the set of tasks and E is the set of edges. The edge (i, k) on the graph specifies that there is some required data transfer, e_{ik}, from task i to task k and hence, k cannot start before i finishes. Scheduling DAG applications consists of two steps. First,

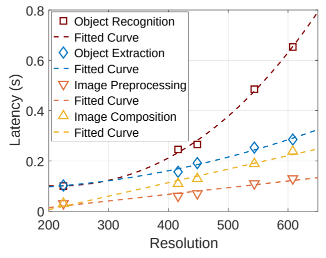


Fig. 14. The relationship between datasize and computation complexity for AR-ralated operations.

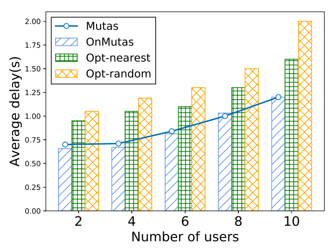


Fig. 15. Impact of the number of users on the performance in terms of average service delay.

at the beginning of each slot, we find all the task nodes with zero in-degree then a scheduling decision is made for them. When these tasks finish, the DAG is updated and zero in-degree tasks can join the queue waiting to be executed. By this way, we decompose sophisticated applications into several waves of tasks and thus they can be successfully scheduled just as pipelines.

Beyond MAR. In this paper, we aim to develop an effective scheduler for the edge-based mobile AR system. However, our system can be easily extended to support heterogeneous applications. For example, the video analysis application can be integrated to our system since they are naturally composed of different computing-intensive components. However, the profiling procedure should be specified for each application due to the inner-application and inter-application diversity.

9 Conclusion

In this paper, a MAR application is modeled as a chain of inter-dependent tasks with data transmissions between them, each task can be offloaded to different servers. We hence consider the offloading of MAR applications comprising multiple tasks, over a generic cloud-edge computing system including a group of heterogeneous edge servers and remote clouds. We design both offline and online algorithms by optimizing the server assignment and the resource allocation jointly. The experimental results show that both our offline and online algorithms can significantly reduce the service delays of MAR applications when compared to other heuristics.

Acknowledgments

This work was supported in part by National Key R&D Program of China (2018YFB1004704), NSFC (61872175, 61832008), Natural Science Foundation of Jiangsu Province (BK20181252), Jiangsu Key R&D Program (BE2018116), and Collaborative Innovation Center of Novel Software Technology and Industrialization. Sheng Zhang is the corresponding author.

References

[1] A. Henrysson and M. Ollila, "Umar: Ubiquitous mobile augmented reality," in *Proc. of ACM Mobiquitous*, 2004, pp. 41–45.

[2] D. Chatzopoulos, C. Bermejo, Z. Huang, and P. Hui, "Mobile augmented reality survey: From where we are to where we go," *IEEE Access*, vol. 5, pp. 6917–6950, 2017.

[3] Q. Liu, J. O. Huang, Siqi, and T. Han, "An edge network orchestrator for mobile augmented reality," in *Proc. of IEEE INFOCOM*, 2018, pp. 1–9.

[4] L. N. Huynh, Y. Lee, and R. K. Balan, "Deepmon: Mobile gpu-based deep learning framework for continuous vision applications," in *Proc. of ACM MobiSys*, 2017, pp. 82–95.

[5] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[6] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computinga key technology towards 5G," *ETSI white paper*, vol. 11, no. 11, pp. 1–16, 2015.

[7] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman, "Live video analytics at scale with approximation and delay-tolerance." in *Proc. of NSDI*, 2017, pp. 1–14.

[8] R. Urgaonkar, S. Wang, T. He, M. Zafer, K. Chan, and K. K. Leung, "Dynamic service migration and workload scheduling in edge-clouds," *Performance Evaluation*, vol. 91, pp. 205–228, 2015.

[9] S. E. Mahmoodi, R. Uma, and K. Subbalakshmi, "Optimal joint scheduling and cloud offloading for mobile applications," *IEEE Transactions on Cloud Computing*, vol. 7, no. 2, pp. 301–313, 2019.

[10] Y.-H. Kao and B. Krishnamachari, "Optimizing mobile computational offloading with delay constraints," in *Proc. of IEEE GLOBECOM*, 2014, pp. 2289–2294.

[11] Y. Zhang, H. Liu, L. Jiao, and X. Fu, "To offload or not to offload: an efficient code partition algorithm for mobile cloud computing," in *Proc. of IEEE CloudNet*, 2012, pp. 80–86.

[12] W. Zhang, Y. Wen, and D. O. Wu, "Energy-efficient scheduling policy for collaborative execution in mobile cloud computing," in *Proc of IEEE INFOCOM*, 2013, pp. 190–194.

[13] X. Zhang, A. Kunjithapatham, S. Jeong, and S. Gibbs, "Towards an elastic application model for augmenting the computing capabilities of mobile devices with cloud computing," *Mobile Networks and Applications*, vol. 16, no. 3, pp. 270–284, 2011.

[14] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, no. 5, pp. 2795–2808, 2016.

[15] B. Cici, M. Gjoka, A. Markopoulou, and C. T. Butts, "On the decomposition of cell phone activity patterns and their connection with urban ecology," in *Proc. of ACM MobiHoc*, 2015, pp. 317–326.

[16] Y.-H. Kao, B. Krishnamachari, M.-R. Ra, and F. Bai, "Hermes: Latency optimal task assignment for resource-constrained mobile computing," *IEEE Transactions on Mobile Computing*, vol. 16, no. 11, pp. 3056–3069, 2017.

[17] M. Jia, J. Cao, and L. Yang, "Heuristic offloading of concurrent tasks for computation-intensive applications in mobile cloud computing," in *Proc. of IEEE INFOCOM WKSHPS*, 2014, pp. 352–357.

[18] M.-A. H. Abdel-Jabbar, I. Kacem, and S. Martin, "Unrelated parallel machines with precedence constraints: application to cloud computing," in *Proc. of IEEE CloudNet*, 2014, pp. 438–442.

[19] S. Sundar and B. Liang, "Offloading dependent tasks with communication delay and deadline constraint," in *Proc. of IEEE INFOCOM*, 2018, pp. 1–9.

[20] K. Habak, M. Ammar, K. A. Harras, and E. Zegura, "Femto clouds: Leveraging mobile devices to provide cloud service at the edge," in *Proc. of IEEE Cloud*, 2015, pp. 9–16.

[21] H. Topcuoglu, S. Hariri, and M.-y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, 2002.

[22] S. Wang, M. Zafer, and K. K. Leung, "Online placement of multi-component applications in edge computing environments," *arXiv preprint arXiv:1605.08023*, 2016.

[23] 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, 2015.

[24] H. Tan, Z. Han, X.-Y. Li, and F. C. Lau, "Online job dispatching and scheduling in edge-clouds," in *Proc. of IEEE INFOCOM*, 2017, pp. 1–9.

[25] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[26] L. Grippo and M. Sciandrone, "On the convergence of the block nonlinear gauss–seidel method under convex constraints," *Operations research letters*, vol. 26, no. 3, pp. 127–136, 2000.

[27] T. L. Beck A, "On the convergence of block coordinate descent type methods," *SIAM journal on Optimization*, vol. 23, no. 4, pp. 2037–2060, 2013.

[28] C. Wang, S. Zhang, H. Zhang, Z. Qian, and S. Lu, "Edge cloud capacity allocation for low delay computing on mobile devices," in *Proc. of IEEE ISPA/IUCC*, 2017, pp. 290–297.

[29] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper, "Workload analysis and demand prediction of enterprise data center applications," in *Proc. of IEEE IISWC*, 2007, pp. 171–180.

[30] M. Jia and J. Cao, "Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks," in *Proc. of IEEE INFOCOM*, 2016, pp. 1–9.

[31] L. Tawalbeh, Y. Jararweh, and F. Ababneh, "Large scale cloudlets deployment for efficient mobile cloud computing," *Journal of Networks*, vol. 10, pp. 70–76, 2015.

[32] Z. Zhou and X. Chen, "Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 36, pp. 2333–2345, 2018.

[33] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *Proc. of IEEE INFOCOM*, 2016, pp. 1–9.

[34] S. Kirkpatrick, "Optimization by simulated annealing: Quantitative studies," *Journal of statistical physics*, vol. 34, no. 5-6, pp. 975–986, 1984.

[35] C. Wang, S. Zhang, Y. Chen, Z. Qian, J. Wu, and M. Xiao, "Joint configuration adaptation and bandwidth allocation for edge-based real-time video analytics," in *Proc. of IEEE INFOCOM*, 2020.

[36] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui, "Visual object tracking using adaptive correlation filters," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE, 2010, pp. 2544–2550.

[37] L. Meylan and S. Susstrunk, "High dynamic range image rendering with a retinex-based adaptive filter," *IEEE Transactions on Image Processing*, vol. 15, no. 9, pp. 2820–2830, 2006.

[38] M. Nixon and A. Aguado, *Feature extraction and image processing for computer vision*. Academic press, 2019.

[39] M. A. Brown, "Traffic control howto," *Guide to IP Layer Network*, p. 49, 2006.

[40] G. Bradski, "The opencv library," http://opencv.org, doctor Dobbs Journal.

[41] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," *arXiv preprint*, 2017.

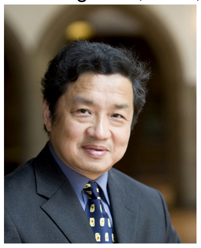
[42] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li, "Lavea: Latency-aware video analytics on edge computing platform," in *Proc. of ACM/IEEE SEC*, 2017.

Can Wang received the BS degree in the School of Computer Science, WuHan University. She is currently working toward the MS degree in the Department of Computer Science and Technology, Nanjing University, under the supervision of Prof. Sheng Zhang. Her research interests include edge computing and wireless networks.

Sheng Zhang is an associate professor in the Department of Computer Science and Technology, Nanjing University. He is also a member of the State Key Lab. for Novel Software Technology. He received the BS and PhD degrees from Nanjing University in 2008 and 2014, respectively. His research interests include cloud computing and edge computing. To date, he has published more than 70 papers, including those appeared in *TMC*, *TPDS*, *TC*, *MobiHoc*, *ICDCS*, *INFOCOM*, *IWQoS*, and *ICPP*. He received the Best Paper Runner-Up Award from IEEE MASS 2012. He is the recipient of the 2015 ACM China Doctoral Dissertation Nomination Award. He is a member of the IEEE and a senior member of the CCF.

Zhuzhong Qian is an associate professor at the Department of Computer Science and Technology, Nanjing University, P. R. China. He received his PhD. Degree in computer science in 2007. Currently, his research interests include cloud computing, distributed systems, and pervasive computing. He is the chief member of several national research projects on cloud computing and pervasive computing. He has published more than 30 research papers in related fields.

Mingjun Xiao is an associate professor in the School of Computer Science and Technology at the University of Science and Technology of China (USTC). He received his Ph.D. from USTC in 2004. In 2012, he was a visiting scholar at Temple University, under the supervision of Dr. Jie Wu. He is a TPC member of many conferences, including IEEE INFOCOM 2018, IEEE ICDCS 2015, ACM Mobihoc 2014, etc, and has served as a reviewer for many journal papers. His main research interests include mobile crowdsensing, mobile social networks, and vehicular ad hoc networks. He has published over 50 papers in refereed journals and conferences, including TON, TMC, TPDS, TC, INFOCOM, etc.



Jie Wu (F'09) is the Director of the Center for Networked Computing and Laura H. Carnell professor at Temple University. He also serves as the Director of International Affairs at College of Science and Technology. He served as Chair of Department of Computer and Information Sciences from the summer of 2009 to the summer of 2016 and Associate Vice Provost for International Affairs from the fall of 2015 to the summer of 2017. Prior to joining Temple University, he was a program director at the National Science Foundation and was a distinguished professor at Florida Atlantic University. His current research interests include mobile computing and wireless networks, routing protocols, cloud and green computing, network trust and security, and social network applications. Dr. Wu regularly publishes in scholarly journals, conference proceedings, and books. He serves on several editorial boards, including IEEE Transactions on Mobile Computing, IEEE Transactions on Service Computing, Journal of Parallel and Distributed Computing, and Journal of Computer Science and Technology. Dr. Wu was general co-chair for IEEE MASS 2006, IEEE IPDPS 2008, IEEE ICDCS 2013, ACM MobiHoc 2014, ICPP 2016, and IEEE CNS 2016, as well as program co-chair for IEEE INFOCOM 2011 and CCF CNCC 2013. He was an IEEE Computer Society Distinguished Visitor, ACM Distinguished Speaker, and chair for the IEEE Technical Committee on Distributed Processing (TCDP). Dr. Wu is a CCF Distinguished Speaker and a Fellow of the IEEE. He is the recipient of the 2011 China Computer Federation (CCF) Overseas Outstanding Achievement Award.

Baoliu Ye is a full professor at Department of Computer Science and Technology, Nanjing University. He received his Ph.D. in computer science from Nanjing University, China in 2004, and was a visiting researcher of the University of Aizu, Japan from March 2005 to July 2006. Currently, he serves as the Dean of School of Computer and Information, Hohai University, China. His current research interests mainly include distributed systems, cloud computing, wireless networks. He has published over 100 papers in referred journals and conferences in above ereas. He is the regent of CCF, the Secretary-General of CCF Technical Committee of Distributed Computing and Systems, and a distinguished member of CCF.

Sanglu Lu received her BS, MS, and PhD degrees from Nanjing University in 1992, 1995, and 1997, respectively, all in computer science. She is currently a professor in the Department of Computer Science and Technology and the State Key Laboratory for Novel Software Technology. Her research interests include distributed computing, wireless networks, and pervasive computing. She has published over 80 papers in referred journals and conferences in the above areas. She is a member of IEEE.