Dynamic Task Scheduling in Cloud-Assisted Mobile Edge Computing

Xiao Ma, Ao Zhou, Shan Zhang, Qing Li, Alex X. Liu, *Fellow, IEEE* and Shangguang Wang, *Senior Member, IEEE,*

Abstract—The cloud-assisted mobile edge computing system is a critical architecture to process computation-intensive and delay-sensitive mobile applications in close proximity to mobile users with high resource efficiency. Due to the heterogenous dynamics of task arrivals at edge nodes and the distributed nature of the system, the workloads of edge nodes are prone to be unbalanced, which can cause high task response time and resource cost. This paper solves the dynamic task scheduling problem in cloud-assisted mobile edge computing (including both peer task scheduling among edge nodes and cross-layer task scheduling from edge nodes to the cloud), aiming at minimizing average task response time within resource budget limit. To overcome the challenges of task arrival dynamics, edge node heterogeneity, and computation-communication delay tradeoff, we propose a Water-filling Based Dynamic Task Scheduling (WiDaS) algorithm. WiDaS dynamically tunes the usage of cloud resources based on the Lyapunov optimization method and efficiently schedules mobile tasks among edge nodes (and the cloud) by exploiting the idea of water filling. Extensive simulations are conducted to evaluate WiDaS under a trace-driven traffic pattern and two mathematic traffic patterns. The results demonstrate that WiDaS shows two-fold benefits of efficiency and effectiveness. In terms of efficiency, WiDaS can achieve the approximate results with the KKT-based algorithm while reducing the computation complexity from exponential order to polynomial order. In terms of effectiveness, WiDaS can reduce the average task response time by up to 64.4% and 47.2% over the Fair-ratio and the Edge-first algorithm.

Index Terms—Mobile edge computing, cloud, task scheduling, workload scheduling

1 Introduction

With the proliferation of computation-intensive and delay-sensitive mobile applications, mobile devices are required to provide enhanced computation capabilities and long-lasting battery life. However, the capacities of mobile devices are inherently limited due to its physical size. Mobile cloud computing has been proposed to augment the capacities of mobile devices by offloading mobile tasks to the configurable and sharable resource pool of cloud computing [1], [2]. Nevertheless, offloading mobile tasks to the remote cloud suffers from wide area network delay and jitters. Moreover, the demands of mobile applications are increasing much faster than the growth rate of core network capacity. All these limitations incent the emergence of mobile edge computing (MEC). By deploying powerful servers (i.e., edge nodes) within the wireless access network, mobile devices are able to access sufficient computation resources via only one-hop wireless transmission, benefiting from reduced delay and energy consumption [3], [4], [5]. Additionally, bandwidth-intensive tasks from mobile devices can be aggregated at the edge nodes, mitigating the burden of core network.

Xiao Ma, Ao Zhou, Qing Li and Shangguang Wang are with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China, 100876. E-mail: maxiao18@bupt.edu.cn, aozhou@bupt.edu.cn, q_li@bupt.edu.cn, sgwang@bupt.edu.cn.
Shan Zhang is with the School of Computer Science and Engineering, Beihang University, Beijing, China, 100191. E-mail: zhangshan18@buaa.edu.cn.
Alex X. Liu is with the Department of Computer Science and Engineering, Michigan State University, East Lansing, MI, USA, 48824-1266. Email: alexliu@cse.msu.edu.

In mobile edge computing, while the computation capacity of each edge node is fixed and limited, mobile task arrivals fluctuate significantly with time [6]. To accommodate the dynamic mobile tasks with high resource efficiency, we exploit the potential of the cloud-assisted mobile edge computing system in resource agility [7]. In this system, cloud resources are used on demand to process the excessive tasks outsourced from edge nodes. Thus, overloaded tasks at each edge node can either be outsourced to the cloud through the core network or migrated to nearby lightloaded edge nodes via the local area network (LAN) or wired peer-to-peer (P2P) connections [8]. Due to the heterogenous dynamics of task arrivals at edge nodes and the distributed nature of the system, the workloads of edge nodes are prone to be unbalanced, resulting in high task response time and resource cost.

According to above analysis, dynamically scheduling mobile tasks to proper edge nodes (or the cloud), which thereby balances the system workloads is crucial for improving the system performance at low resource cost. In this paper, we investigate the dynamic task scheduling issue based on the ETSI MEC architecture [9]. In this architecture, a MEC application instantiation is triggered once an instantiation request is raised from a mobile device or from the Operations Support System. The MEC Orchestration (MEO) selects the ideal MEC server for the MEC application instance, and then routes the corresponding mobile traffic to the selected MEC server. With the above functionality of MEO, we seek to design a dynamic task scheduling strategy, which adaptively guides the MEO to make proper decisions for MEC application instantiation and traffic scheduling.

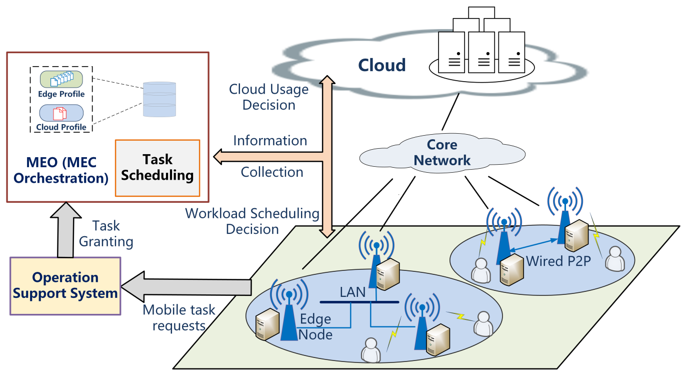


Fig. 1. Task scheduling in cloud-assisted mobile edge computing.

Dynamically scheduling mobile tasks in the cloud-assisted mobile edge computing is a two-fold problem which needs to (i) adaptively tune the usage of cloud resources to cater for time-varying mobile tasks, (ii) properly schedule mobile tasks among edge nodes and the cloud. Solving this problem optimally faces critical challenges. First, the computation capacities of edge nodes are fixed and limited, while mobile task arrivals vary with time significantly due to the mobility of mobile users and fluctuation of wireless environment. Dynamic task scheduling is demanded to process the mobile tasks with reduced delay and resource consumption. Second, edge nodes are heterogeneous in terms of computation capacities and mobile task arrivals. Proper task scheduling requires workload balancing among the heterogeneous edge nodes and the cloud, causing exponential computation complexity. Third, proper task scheduling calls for tradeoff between computation and communication delay. Migrating tasks among nearby edge nodes or outsourcing excessive tasks to the cloud is beneficial to reduce the computation delay. Nevertheless, additional transmission requests are induced on the LAN or the core network, causing extra communication delay. Scheduling mobile tasks among the edge nodes and the cloud should coordinate the computation and the introduced communication delay to optimize the average task response time.

To address the above challenges, we formulate it as a dynamic optimization problem and design an efficient task scheduling algorithm based on the Lyapunov optimization method and the idea of water filling. To deal with the first challenge of task arrival dynamics, we use the Lyapunov optimization method [10] to dynamically tune the usage of cloud resources and cater for dynamic task arrivals with high resource agility. To address the second challenge of exponential complexity caused by edge heterogeneity, we propose the W̲ater-fi̲lling Based D̲ynamic Ta̲sk S̲cheduling (WiDaS) algorithm, which is proved to have polynomial complexity.

To address the third challenge of computation-communication tradeoff, we solve dynamic task scheduling considering the non-negligible transmission delay on LAN. A queuing network is used to model the computing processes at edge nodes and the cloud and the transmitting processes on the LAN and the core network. The computation and communication delay can be properly traded off when the average task response time in the queuing network is minimized.

The contributions of this paper are summarized as follows.

- This work investigates the dynamic task scheduling issue in cloud-assisted mobile edge computing, which is the first to include both peer task scheduling among edge nodes and cross-layer task scheduling from edge nodes to the cloud.
- We formulate it as a dynamic optimization problem and propose the WiDaS algorithm to deal with the challenges of task arrival dynamics, edge node heterogeneity and computation-communication delay tradeoff.
- We conduct extensive simulations to evaluate the WiDaS algorithm under one trace-driven traffic pattern and two mathematic traffic patterns. The simulation results demonstrate that WiDaS shows two-fold benefits of efficiency and effectiveness.

The organization of this paper is as follows. We first review the related work in Sec. 2. In Sec. 3, we first introduce the overview of the system, and then present the analytical model and problem formulation. Sec. 4 provides the problem transformation and performance analysis. Based on the above result, we present the WiDaS algorithm design in Sec. 5. Sec. 6 conducts extensive simulations to evaluate the proposed algorithm, and Sec. 7 concludes the paper.

2 Related Work

To meet the QoS requirements of the delay-sensitive and resource-intensive mobile applications, researchers have devoted extensive efforts to exploiting the potential of mobile edge computing, such as edge-based augmented reality applications [11], edge caching [12], [13], edge-based vehicular network [14], etc. Due to the limited computation capacities of edge nodes and the transmission bandwidth of wireless access network, a lot of work has investigated task scheduling in mobile edge computing. Sundar et al [15] have solved the task scheduling decision problem from the perspective of applications, with each application consisting of multiple dependent tasks which should be processed within a deadline.

There are also extensive works on task scheduling from the perspective of edge operators to provision improved system performance with reduced resource consumption. The existing work on task scheduling in mobile edge computing can be divided into two categories: cross-layer task scheduling and peer task scheduling.

Cross-Layer Task Scheduling: Researchers have investigated cross-layer (device-to-edge or edge-to-cloud) task scheduling to enhance local computation capabilities by solving the computation offloading problem. Distributed computation offloading schemes have been widely investigated in which computation offloading decisions are made in a distributed manner to optimize various objectives, including energy consumption [16], [17], [18], processing delay [19], [20], [21], bandwidth cost [22] and mobility management [23], [24]. Centralized computation scheduling mechanisms have also been proposed by researchers.

Sardellitti *et al.* [25] have jointly optimized the radio and computational resources in multicell mobile edge computing by designing algorithms for both the single-user and multi-user scenarios. Yang *et al.* in [26] have investigated network-aware computation partitioning for multi-user edge computing. Li *et al.* in [27] have designed offloading strategy between edge servers and cloud for deep learning based IoT applications. Kwak *et al.* [28] have proposed an energy-aware dynamic resource and task allocation algorithm to optimize CPU and network energy within delay constraints. In [29], centralized workload scheduling among multiple mobile devices and one single edge node has been investigated. An efficient algorithm has been proposed to achieve centralized access control of mobile tasks from mobile devices to the edge node and proper workload scheduling between the edge node and the cloud. In [30], Xu *et al.* have explored the edge-cloud cooperation to jointly optimize service caching and task offloading. Cloud servers act as a powerful backup resource provider in the system, provisioning ample storage and computation capacity for excessive tasks from each edge node.

Peer Task Scheduling: Researchers have also devoted extensive efforts to peer task scheduling to explore the potential of peer cooperation in improving resource efficiency. Chen *et al.* [8] have exploited cooperation among small base stations (SBSs) to efficiently handle spatially uneven workloads of different SBSs. An online peer offloading algorithm has been developed by leveraging the Lyapunov optimization method to optimize the long-term system performance without knowledge of future information. Cui *et al.* [31] have promoted software defined cooperative offloading in D2D network and scheduled tasks among mobile devices in a centralized manner. A greedy algorithm has been designed to efficiently address the problem with large scale, targeting at reducing the energy consumption of mobile devices and the traffic on the access links.

There are only a few works studying both cross-layer task scheduling and peer task scheduling [32], [33], [34]. Champati et al [32] have solved the task scheduling problem, which decides task offloading to the cloud and task scheduling among local processors, without assuming that task processing times are known a prior. However, the main focus of the work is static task scheduling, and in the dynamic task arrival scenario, the performance bound can only be ensured for special cases (e.g., when the cloud processing delay is negligible). In work [33], [34], dynamic scaling of cloud resources is not considered, which cannot fully improve resource scalability by exploring the resource elasticity of cloud. Our main focus is dynamic task scheduling in the cloud assisted mobile edge computing, including both cross-layer task scheduling and peer task scheduling. Cloud resource usage is dynamically tuned and task scheduling is thereby adjusted among edge nodes and from edge nodes to cloud. An efficient workload scheduling algorithm which achieves close-to-optimal performance (illustrated in Sec. 6.2) with reduced complexity is provided considering the non-negligible transmission delay to the cloud and computation delay in the cloud.

3 System Model and Problem Formulation

In this section, we first introduce the overview of task scheduling in mobile edge computing, then we present the analytical model of the dynamic task scheduling problem. Finally, we provide the problem formulation.

3.1 Task Scheduling in Cloud-Assisted Mobile Edge Computing

In mobile edge computing, neighboring edge nodes are connected by LAN or wired P2P connections. Fig. 1 illustrates the architecture of cloud-assisted mobile edge computing based on the ETSI MEC architecture in [9]. Operation Support System receives mobile requests and makes granting decisions. The granted requests are directed to MEO for request scheduling. To properly scheduling mobile tasks among edge nodes, the MEO periodically collects information from edge nodes, including mobile task arrival rates and available edge resources. With the above information, MEO forms a network-wide view and manages the system in a centralized manner.

Notice that information exchange overhead is inevitable for almost all centralized schemes, consuming communication resources and incurring additional information exchange delay. Nevertheless, the transmitted data size during the information exchange between MEO and edge nodes is quite small compared to the mobile tasks. Thus, the resource consumption and information exchange delay can be considered negligible. Moreover, we seek to design a workload scheduling algorithm that can be adjusted according to the practical scenario. For example, when it is not allowed to schedule tasks among edge nodes across different cities, each city can be managed by a MEO (which can be operated on an edge node). The information exchange delay between the edge nodes and MEO can be as low as 10ms according to our first measurement study on a leading public edge platform in China [35] (even in the worst case, the information exchange delay will not exceed 20ms), which is negligible compared to the task processing delay.

In the cloud-assisted mobile edge computing system, mobile task arrivals at each edge nodes vary with time significantly while the resource capacities of edge nodes are fixed and remain unchanged. To optimize system performance with reduced system resource cost, we investigate dynamic task scheduling for the MEO in cloud-assisted mobile edge computing. To better characterize system dynamics, the task scheduling decision is operated in a slotted manner. In each time slot, MEO dynamically schedules mobile tasks as follows. 1) MEO updates the cloud information of the last time slot kept in the cloud profile table, which records cloud resource utilization, core network bandwidth, and cloud resource cost. 2) MEO updates history information of edge nodes kept in the edge profile tables at MEO, which record mobile task arrival rates and available edge resources. 3) Based on the history information of the edge profile tables, mobile task arrivals of the current time slot can be predicted. This prediction is for short-term instantaneous task arrival of the current time slot, and high prediction accuracy can be achieved with the well-studied prediction methods, such as FFT [36], Vector Autoregression [37], and LSTM [38]. As task arrival prediction is not the main focus of this work, we do not devote efforts to task arrival prediction and provide no results of the prediction. In Sec. 6, we present different task arrival patterns as the prediction results of mobile task arrivals. 4) With the above information, MEO makes the task scheduling decision, including i) cloud usage decision, i.e., the optimal number of the tenanted cloud instances and the optimal service level of core network, ii) edge workload scheduling decision, i.e., the optimal workloads scheduled to edge nodes and the cloud. Through dynamically tuning the usage of cloud resources and thereby adjusting the task scheduling decisions with the above steps, MEO seeks to optimize the long-term system performance within the resource budget limit.

3.2 Analytical Model

This section presents the analytical model of dynamic task scheduling in the cloud-assisted mobile edge computing system. We consider $\mathbb{N} = \{1, 2, ...N\}$ edge nodes that are connected by LAN. The MEC application instantiation and traffic scheduling of the edge nodes are managed by the MEO. Let Υ_n denote the set of neighbouring edge nodes that have connection with edge node n. Each edge node n has fixed and limited computation capability δ_n (in CPU cycle per second). Cloud resources are utilized on demand to meet the QoS requirements of dynamic tasks. In public clouds, users are allowed to scale cloud resource usage by services such as AWS Auto Scaling [39]. We define a decision round as a time slot. At the beginning of each time slot, the controller makes a centralized cloud resource scaling and workload scheduling decision. As the scaling actions can not happen too frequently, a time slot is much longer than the time interval between two successive task arrivals at each edge node. In this work, we consider a set of $\mathbb{T} = \{1, 2, ...T\}$ time slots.

3.2.1 Dynamic Task Arrival

To characterize the dynamics of mobile task arrival at each edge node, we use the non-homogeneous Poisson process to model the mobile task arrival [8], [40]. With this model, the task arrival process at edge node n ($n \in \mathbb{N}$) is a Poisson process in each time slot. Let $A_n(t)$ denote the average arrival rate of mobile tasks at edge node n in time slot t and $A_n(t)$ varies in different time slots. As different mobile tasks have various computation (in CPU cycles) and transmission requests (in bytes), we assume that the required CPU cycles of mobile tasks follow exponential distribution [8], with ξ representing the expectation. When transmitting one unit of mobile tasks (in CPU cycles), c units of input data are required (in bytes).

3.2.2 Executing at Edge Nodes

As edge nodes have heterogeneous computation capacities and uneven mobile task arrivals, migrating tasks among nearby edge nodes is beneficial to exploit the under-utilized edge nodes and balance edge workloads. According to mobile task arrivals and the available edge computation capacities, the edge nodes can be divided into three categories: overloaded edge nodes, neutral edge nodes, and light-loaded edge nodes: 1) For a overloaded edge nodes s, a fraction of tasks are migrated to nearby edge nodes or outsourced to the cloud. Denote by $r_{sd}(t)$ the fraction of tasks migrated from edge node s to edge node d in time t ($r_{sd}(t) \geq 0$). In particular, $r_{ss}(t)$ represents the fraction of tasks executed locally, and $r_{s0}(t)$ represents the fraction of tasks outsourced to the cloud. Thus, $r_{sd}(t)$ satisfies $\sum_{d \in N \cup \{0\}} r_{sd}(t) = 1$ and the tasks executed at edge node s follow a Poisson process with the parameter $\lambda_s(t)$ given as $\lambda_s(t) = r_{ss}(t) \cdot A_s(t)$. 2) For a neutral edge node m, it neither receives tasks from nearby edge nodes ($r_{sm} = 0$) nor migrates tasks to other edge nodes ($r_{md} = 0$) or the cloud

TABLE 1
Table of Notations

Notation	Description
Υ_n	Set of Neighbouring nodes of edge node n
δ_n	Computation capacity of edge node n
$A_n(t)$	Average task arrival rate at edge node n in tth slot
ξ	Expected CPU cycles of mobile tasks
c	Number of input data required by one unit of tasks
$r_{\text{sd}}(t)$	Fraction of tasks migrated from edge node s to d
$r_{\text{s0}}(t)$	Fraction of tasks outsourced to the cloud from s
$\lambda_n(t)$	Expected rate of tasks processed at edge node n
$\lambda_{\text{out}}(t)$	Expected rate of outsourced tasks to the cloud
$\lambda_{\text{cloud}}(t)$	Expected task arrival rate in the cloud
τ_n	Expected serving time of tasks at edge node n
μ_n	Expected serving rate of edge node n
$\mu_{\text{core}}(t)$	Expected serving rate of the core network
$\mu_{\text{c}}(t)$	Expected serving rate of the cloud
$\lambda_{\cdot n}(t)$	Expected inbound rate at n from other edge nodes.
$x(t)$	Service level of AWS Direct Connect
$b_{\text{core}}(t)$	Transmission rate of the core network
$A(t)$	Overall task arrival rates at edge nodes in t
$d(t)$	LAN congestion delay at edge node n
Θ	Resource budget limit of the edge operator
$\hat{s}(t)$	Task scheduling decision in t
$Q(t)$	Virtual queue backlog in t
V	Cost-performance tradeoff
$L(t)$	Lyapunov function in t
$\Delta(Q(t))$	Lyapunov drift in t
η_j, σ_i	Lagrange multipliers, ($j = 1, 2$), ($i = 1, 2, ..., 2N + 4$)

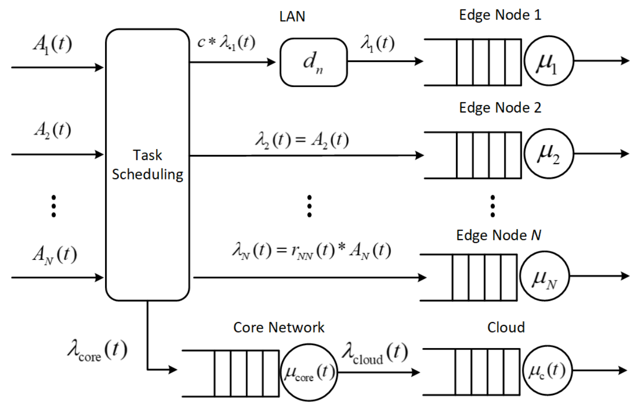


Fig. 2. Queuing network model.

($r_{m0} = 0$). Thus, the tasks executed at edge node m follow a Poisson process with the parameter of $A_m(t)$ (i.e. $r_{mm}(t) = 1$, $\lambda_m(t) = A_m(t)$). 3) For a light-loaded edge node d, the tasks include the tasks originally arriving at this edge node and those migrated from the overloaded edge nodes. Thus the tasks executed at edge node d follow a Poisson process (sum of several Poisson processes) and the parameter $\lambda_d(t)$ is given as $\lambda_d(t) = A_d(t) + \sum\limits_{s\in N\setminus\{d\}} r_{sd}(t)A_s(t)$.

Computation Delay: By summarizing above analysis, the task arrival at each edge node can be modeled as a Poisson process in each time slot. In addition, as the required computation of different tasks follows exponential distribution with the parameter of ξ, the serving time at each edge node also follows exponential distribution with the expectation

$$\tau_n = \frac{\xi}{\delta_n}. \tag{1}$$

Therefore, in each time slot t, the serving process at each edge node can be modeled as an M/M/1 queue and the computation delay is given as

$$D_n(t) = \frac{1}{\mu_n - \lambda_n(t)}, \tag{2}$$

where $\mu_n = \frac{1}{\tau_n}$.
To ensure the stability of the queue, it should be satisfied that $\lambda_n(t) < \mu_n$ for any time t. Thus, $\lambda_n(t)$ is constrained as

$$0 \leq \lambda_n(t) < \mu_n. \tag{3}$$

Moreover, as only the neighbouring edge nodes in Υ_n can migrate tasks to edge node n, the tasks executed at n cannot exceed the overall tasks arriving at the neighbouring edge nodes. Therefore, by combining the queue stability condition in Eq. (3), $\lambda_n(t)$ should satisfy

$$0 \leq \lambda_n(t) < \min\{\sum_{i\in\Upsilon_n\cup\{n\}} A_n(t), \mu_n\}. \tag{4}$$

Migration Delay among Edge Nodes: Migrating mobile tasks among edge nodes can cause additional transmission requests on LAN. Let $d_n(t)$ denote the migration delay of inbound tasks to edge node n. We consider that neighboring edge nodes are connected by high-bandwidth wired connections, and the migration delay $d_n(t)$ ($d_n(t) \geq 0$) is irrelevant to the transmitted data size. Denote by $\lambda_{\cdot n}(t)$ the expected rate of incoming tasks at edge node n from other edge nodes. Thus, $\lambda_{\cdot n}(t)$ can be given as $\lambda_{\cdot n}(t) = \sum\limits_{s\in N\setminus\{n\}} r_{sn}(t)\lambda_s(t)$, and it is satisfied that

$$\lambda_{\cdot n}(t) = \max\{\lambda_n(t) - A_n(t), 0\}. \tag{5}$$

3.2.3 Outsourcing to the Cloud

Public clouds provide users with elastic computation resources that are encapsuled as cloud instances and charged based on usage hourly or even by second [41]. To accommodate dynamic mobile tasks with high resource efficiency, the controller tunes the usage of cloud instances and outsources the excessive tasks to the cloud at peak hours.

When outsourcing workloads to the cloud, input data of the tasks needs to be transmitted through the core network. Since wide area network delay and jitters can be induced during the transmission [42], a dedicated connection should be established between edge nodes and the cloud to guarantee the system performance. Amazon makes it possible by providing the service AWS Direct Connect [43], enabling users to select from different service levels $\mathbb{L} = \{0, 1, ..., L\}$. The transmission rate and pricing rate of AWS Direct Connect are determined by the service level $x(t)$, denoted as $R(x(t))$ and $P(x(t))$ respectively. Note that $x(t) = 0$ indicates that no data is transmitted through the core network, i.e. the transmission rate $R(0) = 0$, and the pricing rate $P(0) = 0$. Table 2 lists the transmission rates and pricing rates of different service levels.

TABLE 2
Pricing of AWS Direct Connect

Service level	1	2	3	4	5	6	7	8
Bandwidth (Mbps)	50	100	200	300	400	500	10^3	10^4
Pricing rate (USD/h)	0.03	0.06	0.12	0.18	0.24	0.30	0.30	2.25

Denote by $\lambda_{\text{out}}(t)$ the average rate of outsourced tasks to the cloud ($\lambda_{\text{out}}(t) \geq 0$), which can be computed as

$$\lambda_{\text{out}}(t) = \sum_{n=1}^{N} A_n(t) - \sum_{n=1}^{N} \lambda_n(t). \tag{6}$$

Let $b_{\text{core}}(t)$ be the transmission rate of the core network ($b_{\text{core}}(t) = R(x(t))$), then the transmission delay in the core network is

$$D_{\text{core}}(t) = \frac{1}{\mu_{\text{core}}(t) - \lambda_{\text{out}}(t)}, \tag{7}$$

where $\mu_{\text{core}}(t) = \frac{b_{\text{core}}(t)}{c\xi}$. To ensure the stability of data transmission in the core network, it should be ensured that

$$0 \leq \lambda_{\text{out}}(t) < \mu_{\text{core}}(t). \tag{8}$$

Denote by $\lambda_{\text{cloud}}(t)$ ($\lambda_{\text{cloud}}(t) \geq 0$) the task arrival rate in the cloud, there is

$$\lambda_{\text{cloud}}(t) = \lambda_{\text{out}}(t). \tag{9}$$

As cloud resources are provisioned as encapsuled instances, let $m(t)$ represent the number of tenanted cloud instances in time t.

$$0 \leq m(t) \leq m_{\max}, \tag{10}$$

where $m_{\max}$ is the maximum cloud instance number that can be used. In the cloud, the usage of cloud resources is tuned to accommodate dynamic tasks with high resources efficiency. With services such as AWS Auto Scaling, cloud users are allowed to scale cloud resource usage by specifying scaling rules. In this work, the logical controller scales the cloud resource usage (i.e., the number of tenanted cloud instances $m(t)$) according to cloud resource utilization $\rho(t)$:

$$m(t+1) = \begin{cases} m(t) + i, & \rho(t) \geq U_{\text{up}} \\ \max\{m(t) - j, 0\}, & \rho(t) < U_{\text{down}} \\ m(t), & \text{otherwise.} \end{cases} \tag{11}$$

Here, i, j are positive constants, and U_{up}, U_{down} represent the upper and lower bound of the cloud resource utilization. The cloud resource utilization $\rho(t)$ is represented by the resource utilization of the queue in the cloud (the definition is given later). Eq. (11) indicates that the number of tenanted cloud instances increases by i when the cloud resource utilization of last time slot grows higher than the upper bound, and the number decreases by j (if positive) when the cloud resource utilization drops lower than the lower bound.

With the above scaling rules, the computing rate of the cloud can be given as $\delta_{\text{c}}(t) = m(t)\delta_{\text{ins}}$, where δ_{ins} is the computing rate of one cloud instance. The serving time of tasks in the cloud also follows exponential distribution with the expectation $\frac{1}{\mu_{\text{c}}(t)}$. Here, $\mu_{\text{c}}(t)$ is the expected serving rate of the cloud and can be given as $\mu_{\text{c}}(t) = \frac{\delta_{\text{c}}(t)}{\xi}$. Similar as Eq. (2), the serving process at the cloud can be modeled as an M/M/1 queue, and the average computation delay is

$$D_{\text{cloud}}(t) = \frac{1}{\mu_{\text{c}}(t) - \lambda_{\text{cloud}}(t)}, \tag{12}$$

where $\lambda_{\text{cloud}}(t)$ is constrained as

$$0 \leq \lambda_{\text{cloud}}(t) < \mu_{\text{c}}(t). \tag{13}$$

We define the resource utilization of the queue in the cloud as the cloud resource utilization, which is computed as $\rho(t) = \frac{\lambda_{\text{cloud}}(t)}{\mu_{c}(t)}$. Note that the actuation delay, i.e., the start-up time of cloud instances, can be several minutes, which is unacceptable for delay-sensitive mobile applications [44]. To deal with this problem, AWS Auto Scaling service [39] provides the predictive scaling policy to detect cloud usage patterns periodically and perform scaling actions in advance of predicted changes. Therefore, the performance degradation caused by actuation delay can be avoided.

3.3 Problem Formulation

3.3.1 System Cost

In the cloud-assisted mobile edge computing framework, the system cost includes the on-premise cost of edge nodes and the usage-based cost of cloud resources. Let C_{edge} denote the cost of edge nodes, which remains unchanged through time since fixed computation capacities are provisioned at edge nodes. The outsourcing cost of cloud resources consists of the cost on cloud instances and the cost on the core network bandwidth (e.g., AWS Direct Connect). When tenanting $m(t)$ cloud instances and selecting the service level $x(t)$, the outsourcing cost on cloud resources $C_{\text{cloud}}(t)$ is

$$C_{\text{cloud}}(t) = m(t)p_{\text{ins}} + P(x(t)). \tag{14}$$

Here, p_{ins} is the price of one cloud instance.
Hence, the system cost is

$$C(t) = C_{\text{edge}} + C_{\text{cloud}}(t). \tag{15}$$

3.3.2 System Performance

We take the average task response time as the metrics of system performance. Let $A(t)$ denote the overall task arrival rates at all edge nodes in time t, i.e., $A(t) = \sum_{n=1}^{N} A_n(t)$. The average task response time is a weighted sum of delay at each part of Fig. 2 and can be computed as

$$\begin{aligned} D(t) = & \underbrace{\sum_{n=1}^{N} (\frac{\lambda_n(t)}{A(t)} D_n(t) + \frac{\lambda_{\cdot n}(t)}{A(t)} d_n(t))}_{\text{executing at edge nodes}} \\ & \underbrace{+ \frac{\lambda_{\text{out}}}{A(t)} D_{\text{core}}(t) + \frac{\lambda_{\text{cloud}}}{A(t)} D_{\text{cloud}}(t),}_{\text{outsourcing to the cloud}} \end{aligned} \tag{16}$$

As shown in the above equation, the average task response time consists of the weighted delay when executed at edge nodes and the weighted delay when outsourced to the cloud. For the tasks executed at edge nodes, the weighted delay includes the computation delay at edge nodes and the migration delay among edge nodes. Note that only the migrated tasks from other edge nodes can induce transmission delay at edge node n. Thus, the weight of the migration delay is $\frac{\lambda_{\cdot n}(t)}{A(t)}$. When outsourcing mobile tasks to the cloud, the delay is composed of the computation delay in the cloud and the transmission delay in the core network.

In the cloud-assisted mobile edge computing system, to optimize the average task response time within the budget limit, the dynamic task scheduling problem is formulated as

$$\begin{aligned} & \textbf{P1: } \min \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} D(t) \\ & s.t. \\ & \textbf{C1: } \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} C(t) \leq \Theta \\ & \text{Constranits (4) (6) (8) (9) (13).} \end{aligned} \tag{17}$$

In problem **P1**, the objective is to optimize the long-term average task response time. Constraint **C1** ensures the long-term average system cost does not exceed the resource budget limit Θ. We seek to determine the optimal cloud resource usage (i.e., the number of cloud instances, $m(t)$, and the service level of the core network, $x(t)$) and the optimal task scheduling decision $\hat{s}(t)$ ($\hat{s}(t) = \langle \lambda_1(t), ..., \lambda_N(t), \lambda_{\text{out}}(t), \lambda_{\text{cloud}}(t) \rangle$) by solving problem **P1**.

4 Problem Transformation and Performance Analysis

In this section, we first present problem transformation based on the Lyapunov optimization method. Then we analyze the performance of this transformation, providing theoretical basis for algorithm design in the next section.

As the usage of cloud resources is tuned dynamically, the system cost in different time slots can exceed or remain under the budget limit. To characterize the relative states of system cost and budget limit, we introduce the virtual queue $Q(t)$ given as

$$Q(t) = \begin{cases} \max\{Q(t-1) + C(t-1) - \Theta, 0\}, & t \geq 1 \\ 0, & t = 0. \end{cases} \tag{18}$$

Eq. (18) indicates that $Q(t)$ accumulates the excessive system cost over the budget limit. With $Q(t)$, we have the following conclusion: *Constraint **C1** can be ensured by enforcing the stability of $Q(t)$.*

Proof. According to the definition of $Q(t)$, the deviation of $Q(t)$ satisfies

$$Q(t+1) - Q(t) \geq C(t) - \Theta. \tag{19}$$

Sum up the deviation through time slots $t \in \mathbb{T}$, divide T and take a limit over T, there is

$$\lim_{T\to\infty} \frac{Q(T) - Q(0)}{T} \geq \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} C(t) - \Theta. \tag{20}$$

By enforcing the stability of $Q(t)$, i.e.,

$$\lim_{T\to\infty} \frac{Q(T)}{T} = 0, \tag{21}$$

we have

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} C(t) - \Theta < 0. \tag{22}$$

Thus the constraint **C1** is ensured. □

According to the above conclusion, it follows that if we can design a dynamic task scheduling algorithm that minimizes the long-term average task response time subject to the stability of virtual queue $Q(t)$, the original problem **P1** is equivalently solved. As the Lypunov optimization method is dedicated to optimizing objectives while maintaining queue stability, this conclusion leads to solving problem **P1** with the Lyapunov optimization method.

4.1 Problem Transformation

To solve **P1** with the Lyapunov optimization method, we first introduce the Lyapunov function and Lyapunov drift. Define the Lyapunov function as

$$L(t) = \frac{1}{2} Q^2(t), \tag{23}$$

and the Lyaponov drift as

$$\Delta(Q(t)) = \mathbb{E}\{L(Q(t+1)) - L(Q(t))|Q(t)\}. \tag{24}$$

It can be inferred from [10] that the smaller is $\Delta(Q(t))$ (for each $t \in \mathbb{T}$), the more likely $Q(t)$ is stabilized. In problem **P1**, apart from the virtual queue $Q(t)$ we want to stabilize, there is an associated "penalty" process $D(t)$, the long-term average of which we want to minimize. Hence, to solve **P1**, we exploit the conclusion of the Lyapunov optimization method and turn to minimize the drift-plus-penalty $\Delta(Q(t)) + V\mathbb{E}\{D(t)|Q(t)\}$ in each $t \in \mathbb{T}$. Here V ($V \geq 0$) implies the cost-performance tradeoff, i.e., how much the algorithm design emphasizes the system performance compared with the system cost.

From the definition of $\Delta(Q(t))$, we can derive that the drift-plus-penalty is upper bounded as

$$\begin{aligned}&\Delta(Q(t)) + V\mathbb{E}\{D(t)|Q(t)\} \leq \\ &B + Q(t)\mathbb{E}\{(C(t) - \Theta)|Q(t)\} + V\mathbb{E}\{D(t)|Q(t)\},\end{aligned} \tag{25}$$

where B is a constant and satisfies

$$B \geq \frac{1}{2}\mathbb{E}\{(C(t) - \Theta)^2|Q(t)\}. \tag{26}$$

for all time t. According to the Lyapunov drift-plus-penalty framework in [10], we just need to optimize the bound in the right side of (25) in each time slot t, then the problem **P1** is transformed into

$$\begin{aligned}\textbf{P2:}\ &\min Q(t)(C(t) - \Theta) + VD(t) \\ &s.t.\ \text{Constraints (4) (6) (9) (8) (13).}\end{aligned} \tag{27}$$

Till now, we have transformed the original dynamic optimization problem **P1** to a static optimization problem **P2** in each time slot. In the following part, we will analyze the performance of this problem transformation.

4.2 Performance Analysis

We present theoretical analysis of the problem transformation, demonstrating that solving **P2** optimally in each time slot can lead to near-optimal solution of problem **P1**. The results are concluded in Theorem 1.

Theorem 1. *Denote by $\hat{s}^*(t)$ the optimal workload scheduling decision obtained by solving problem **P2** in each time slot. The long-term average virtual queue backlog $Q(s^*(t))$ and the service response time $D(s^*(t))$ satisfy*

$$\begin{aligned}&\lim_{T\to\infty} \tfrac{1}{T} \sum_{t=0}^{T-1} E\{D(s^*(t))\} \leq \tfrac{B}{V} + d^* \\ &\lim_{T\to\infty} \tfrac{1}{T} \sum_{t=0}^{T-1} E\{C(s^*(t)) - \Theta\} \leq \tfrac{1}{\epsilon}(B + V(d^{\max} - d^*)),\end{aligned} \tag{28}$$

where d^* is the optimal long-term average service response time to problem P1, $d^{\max}$ is the upper bound of the service response time, and ϵ is the system cost surplus which can be achieved by a stationary workload scheduling strategy.

Proof. From Theorem 4.5 in [10] , for any $\chi > 0$, there exists a stationary workload scheduling policy $\hat{s}^{\Pi}(t)$ for problem **P2**, which is independent of the virtual queue satisfying

$$\begin{aligned}&E\{D(s^{\Pi}(t))\} \leq d^* + \chi, \\ &E\{C(s^{\Pi}(t))\} \leq \chi\end{aligned} \tag{29}$$

By applying Eq. (29) into the drift-plus-penalty inequation (25), there is

$$\begin{aligned}&\Delta(Q(t)) + VE\{D(s^*(t))|Q(t)\} \\ &\leq B + Q(t)E\{(C(s^*(t)) - \Theta)|Q(t)\} + VE\{D(s^*(t))|Q(t)\} \\ &\leq B + Q(t)E\{(C(s^{\Pi}(t)) - \Theta)|Q(t)\} + VE\{D(s^{\Pi}(t))|Q(t)\} \\ &\leq B + \chi Q(t) + V(d^* + \chi)\end{aligned} \tag{30}$$

Let $\chi \to 0$, sum up both sizes of (30) over $t \in \{0, 1, ..., T-1\}$ and divide by T, we obtain

$$\frac{1}{T}E\{L(Q(T)) - L(Q(0))\} + \frac{V}{T}\sum_{t=0}^{T-1} E\{D(s^*(t))\} \leq B + Vd^*. \tag{31}$$

Let $T \to \infty$, (31) is rearranged as

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} E\{D(s^*(t))\} \leq \frac{B}{V} + d^*. \tag{32}$$

To prove (28), we assume that there are $\epsilon > 0$ and a workload scheduling policy $s^w(t)$ satisfying

$$E\{C(s^W(t)) - \Theta\} \leq -\epsilon. \tag{33}$$

Apply (33) to (25), there is

$$\begin{aligned}&\Delta(Q(t)) + VE\{D(s^*(t))|Q(t)\}\\ &\leq B + Q(t)E\{(C(s^W(t)) - \Theta)|Q(t)\} + VE\{D(s^W(t))|Q(t)\}\\ &\leq B - \epsilon Q(t) + VE\{D(s^W(t))\}.\end{aligned} \tag{34}$$

Sum up both sizes of (34) over $t \in \{0, 1, 2, ..., T-1\}$ and divide by T, we have

$$\begin{aligned}&\tfrac{1}{T}E\{L(Q(T)) - L(Q(0))\} + \tfrac{V}{T}\sum_{t=0}^{T-1} E\{D(s^*(t))\}\\ &\leq B - \tfrac{\epsilon}{T}\sum_{t=0}^{T-1} Q(t) + VE\{D(s^W(t))\}\end{aligned} \tag{35}$$

Let $T \to \infty$, and (35) can be rearranged as

$$\begin{aligned}&\lim_{T\to\infty} \tfrac{1}{T}\sum_{t=0}^{T-1} E\{Q(t)\} \leq\\ &\tfrac{1}{\epsilon}(B + \lim_{T\to\infty} \tfrac{V}{T}\sum_{t=0}^{T-1} E\{D(s^W(t))\})\\ &-\tfrac{1}{\epsilon}\lim_{T\to\infty} \tfrac{V}{T}\sum_{t=0}^{T-1} E\{D(s^*(t))\}\\ &\leq \tfrac{1}{\epsilon}(B + V(d^{\max} - d^*))\end{aligned} \tag{36}$$

According to the definition of $Q(t)$, there is

$$\lim_{T\to\infty} \frac{1}{T}\sum_{t=0}^{T-1} E\{C(s^*(t)) - \Theta\} \leq \frac{1}{\cdot}(B + V(d^{\max} - d^*)). \tag{37}$$

By summarizing the above analysis, Theorem 1 can be proved.

□

Insight: Theorem 1 indicates that solving problem **P2** optimally can lead to the near-optimal solution of the o-riginal problem by randomly changing the weight constant V. In each time slot t, observe the current state of $Q(t)$ and solve the static optimization problem **P2**, then the long-term average task response time $D(t)$ will be either smaller than the target value d^* or differs from d^* by less than $\frac{B}{V}$. However, the long-term average virtual queue backlog increases linearly with V, as illustrated in Eq. (28). As the virtual queue accumulates the excessive part of the system cost over the budget limit, Theorem 1 also demonstrates the $[O(V), O(\frac{1}{V})]$ tradeoff between the average task response time and the system cost.

5 The WiDaS Algorithm Design

Sec. 4 has transformed the original dynamic optimization problem **P1** into a static optimization problem **P2** in each time slot. In this section, we first analyze the computation complexity of problem **P2** by exploiting the convex property. Then, we present the algorithm design of WiDaS.

5.1 Complexity Analysis

When given the cloud computation capacity (i.e., the number of cloud instances $m(t)$) and the core network bandwidth (i.e., the service level of AWS Direct Connect $x(t)$), problem **P2** has the the following property.

Theorem 2. *When given $m(t)$ and $x(t)$, problem* **P2** *is a convex optimization problem over the task scheduling decision $\hat{s}(t)$.*

Proof. According to the definition of a convex optimization problem, if we want to prove the covexity of problem **P2**, we need to prove that the objective function $Q(t)(C(t) - \Theta) + VD(t)$ is convex over $\hat{s}(t)$. In each time slot t, the virtual queue $Q(t)$ is updated according to (18) and $C(t)$ is determined by $m(t)$ and $x(t)$. Thus, it is needed to prove that $D(t)$ is convex over $\hat{s}(t)$. According to (16),

$$\begin{aligned}D(t) =& \frac{1}{A(t)}[\sum_{n=1}^{N} \frac{\lambda_n(t)}{\mu_n - \lambda_n(t)} + \lambda_{\cdot n}(t)d_n(t) +\\ &\frac{\lambda_{\text{out}}(t)}{\mu_{\text{core}}(t) - \lambda_{\text{out}}(t)} + \frac{\lambda_{\text{cloud}}(t)}{\mu_{\text{c}}(t) - \lambda_{\text{cloud}}(t)}].\end{aligned} \tag{38}$$

Here, $\lambda_{\cdot n}(t) = \max\{\lambda_n(t) - A_n(t), 0\}$ is convex over $\lambda_n(t)$, and $d_n(t)$ is constant with respect to $\lambda_n(t)$, we just need to prove that $D^{'}(t) = [\sum_{n=1}^{N} \frac{\lambda_n(t)}{\mu_n - \lambda_n(t)} + \frac{\lambda_{\text{out}}(t)}{\mu_{\text{core}}(t) - \lambda_{\text{out}}(t)} + \frac{\lambda_{\text{cloud}}(t)}{\mu_{\text{c}}(t) - \lambda_{\text{cloud}}(t)}]$ is convex over $\lambda_n(t)$. Denote by $H(D^{'}(t))$ the Hessian Matrix of $D(t)$,

$$H(D^{'}(t)) = [h_{mn}]_{(N+2)(N+2)}, \tag{39}$$

and $h_{mn}(t)$ is given as

$$h_{mn}(t) = \begin{cases} 0, & m \neq n\\ \dfrac{2\mu_n}{(\mu_n - \lambda_n(t))^3}, & m = n, n \in \{1, 2, ..., N\}\\ \dfrac{2\mu_{\text{core}}(t)}{(\mu_{\text{core}}(t) - \lambda_{\text{out}}(t))^3}, & m = n = N+1\\ \dfrac{2\mu_c(t)}{(\mu_c(t) - \lambda_{\text{cloud}}(t))^3}, & m = n = N+2.\end{cases} \tag{40}$$

We can conclude that $h_{nn}(t) > 0$ for each $n \in \{1, 2, ..., N+2\}$, and $h_{mn}(t) = 0$ for any $m \neq n$, thus $H(D^{'}(t))$ is a positive semi-definite matrix. Hence, it can be proved that $D^{'}(t)$ is convex over $\hat{s}(t)$ and thus, $D(t)$ is convex over $\hat{s}(t)$.

By summarizing above analysis, the objective function of **P2** is a convex function over $\hat{s}(t)$, the inequation constraints are convex over $\hat{s}(t)$, and the equation constraints are affine functions of $\hat{s}(t)$. Therefore, problem **P2** is a convex optimization problem over $\hat{s}(t)$ [45]. □

Based on Theorem 2, we have transformed the original dynamic optimization problem **P1** to the convex optimization problem **P2** with the Lyapunov optimization method. Note that we remove the time parameter in this section since problem **P2** is a static convex optimization problem. In a convex optimization problem, the points satisfying the KKT conditions are the optimal solutions to this problem [45]. From convex optimization analysis, the KKT conditions of **P2** are given as follows,

$$\begin{aligned}\nabla y(\hat{s}) + \sum_{i=1}^{2N+4} \sigma_i \nabla f_i(\hat{s}) + \sum_{j=1}^{2} \eta_j \nabla g_j(\hat{s}) &= 0\\ \sigma_i f_i(\hat{s}) = 0 \quad & i = 1, 2, ..., 2N+4\\ \sigma_i \geq 0 \quad & i = 1, 2, ..., 2N+4\\ f_i(\hat{s}) \leq 0 \quad & i = 1, 2, ..., 2N+4\\ g_j(\hat{s}) = 0 \quad & j = 1, 2.\end{aligned} \tag{41}$$

Here,

$$y(\hat{s}) = Q(C - \Theta) + VD(\hat{s}), \tag{42}$$

η_j and σ_i are Lagrange multipliers, and $f_i(\hat{s})$ $(i = 1, 2, ..., 2N+4)$ are standard form of inequation constraints in (27), which are defined as

$$f_i(\hat{s}) = \begin{cases} \lambda_i - (\mu_i - \varepsilon), & i = 1, ..., N \\ -\lambda_{i-N}, & i = N+1, ..., 2N \\ \lambda_{\text{out}} - (\mu_{\text{core}} - \varepsilon), & i = 2N+1 \\ -\lambda_{\text{out}}, & i = 2N+2 \\ \lambda_{\text{cloud}} - (\mu_{\text{c}} - \varepsilon), & i = 2N+3 \\ -\lambda_{\text{cloud}}, & i = 2N+4. \end{cases} \tag{43}$$

Here, ε is a positive constant.
$g_j(\hat{s})$ $(j = 1, 2)$ are the standard form of equation constraints in (41) given as

$$\begin{aligned} g_1(\hat{s}) &= \sum_{n=1}^{N} (A_n - \lambda_n) - \lambda_{\text{out}} \\ g_2(\hat{s}) &= \lambda_{\text{cloud}} - \lambda_{\text{out}}. \end{aligned} \tag{44}$$

Insight: The KKT conditions (41) indicate that the optimal solutions of the convex optimization problem are searched among the extreme points and boundary points. For each inequation constraint $f_i(\hat{s})$ $(i = 1, 2, ..., 2N+4)$, there are two possible results: $f_i(\hat{s}) = 0, \sigma_i \geq 0$ implies that the optimal solution is on the boundary; $f_i(\hat{s}) < 0, \sigma_i = 0$ indicates that the optimal solution is at the extreme points. As there are $(2N+4)$ inequation constraints in problem **P2**, directly searching for the points satisfying the KKT conditions can yield $O(2^{2N+4})$ computation complexity. Therefore, solving **P2** based on the KKT conditions has an exponential computation complexity over the number of edge nodes, N. When N is large, a task scheduling algorithm with reduced computation complexity is needed.

5.2 Efficient Task Scheduling Based on Water Filling

With the increasing density of edge nodes in 5G era, N grows rapidly. In this section, we seek to design an efficient task scheduling algorithm with reduced computation complexity.

The complexity analysis in Sec. 5.1 implies that the exponential computation complexity of **P2** arises from the inequation constraints, each of which limits the searching range with the upper and lower bounds for each task scheduling variable in $\hat{s}$. By exploiting this property, we design an efficient task scheduling algorithm as follows. We first assume that each part of the system, including the edge nodes, the core network, and the cloud, has unlimited resource capacity by removing the inequation constraints in **P2**. Then we can compute the relative relationship of the task scheduling variables by solving the modified convex optimization problem **P2** in O(1) computation complexity. Finally, for the task scheduling variables which have grown or dropped beyond the bounds, we take the value of the upper or lower bounds; for the other variables, we still search for the results subject to the above relative relationship. This process is similar to filling water to tubes with different upper bounds. We first assume that the tubes have no upper bounds and fill water to these tubes subject to a certain relative relationship. Once the water level of one tube achieves the upper bound, we will no longer fill water to this tube, and the water level remains at the upper bound. Therefore, our algorithm is called Water-filling Based Dynamic Task Scheduling (WiDaS) algorithm.

The details of the algorithm are as follows:
Step 1. Remove the inequation constraints of **P2**, and search for the extreme points of $y(\hat{s})$ within the equation constraints by solving the modified **P2**.

When removing the inequation constraints of **P2**, the KKT conditions are transformed as:

$$\begin{aligned} \nabla y(\hat{s}) + \sum_{j=1}^{2} \eta_j \nabla g_j(\hat{s}) = 0 \\ g_1(\hat{s}) = 0. \end{aligned} \tag{45}$$

$$g_2(\hat{s}) = 0. \tag{46}$$

Note that $y(\hat{s})$ is not partially derivable over λ_n when $\lambda_n = A_n$ (caused by $\lambda_{\cdot n} = \max\{\lambda_n - A_n\}$ in (16)). This can be solve by dividing λ_n into two cases, i.e., $\lambda_n \geq A_n$ and $\lambda_n < A_n$. Thus combining (45), the workload scheduling variables in $\hat{s}$ can be represented as the functions of η_1:

$$\begin{aligned} \lambda_n(\eta_1) &= \begin{cases} \mu_n - \sqrt{\dfrac{V\mu_n}{A(\eta_1 - d_n)}}, & \eta_1 \geq \beta_n + d_n \\ \mu_n - \sqrt{\dfrac{V\mu_n}{\eta_1 A}}, & 0 \leq \eta_1 < \beta_n \\ A, & \text{otherwise} \end{cases} \\ \lambda_{\text{out}}(\eta_1) &= \mu_{\text{core}} - \sqrt{\tfrac{V\mu_{\text{core}}}{(\eta_1 + \eta_2(\eta_1))A}} \\ \lambda_{\text{cloud}}(\eta_1) &= \mu_c - \sqrt{\tfrac{-V\mu_c}{\eta_2(\eta_1)A}}, \end{aligned} \tag{47}$$

where

$$\beta_n = \frac{V\mu_n}{A(\mu_n - A_n)^2}, \tag{48}$$

$$\eta_2(\eta_1) = \frac{V\mu_{\text{core}}}{A[\mu_{\text{core}} - (A - \sum\limits_{n=1}^{N} \lambda_n)]^2} - \eta_1. \tag{49}$$

Step 2. Restrict these workload scheduling variables within the upper and lower bounds (i.e., inequation constraints of **P2**) as follows:

$$\lambda_n = \begin{cases} 0, & \text{if } \lambda_n(\eta_1) < 0 \\ \mu_n - \varepsilon, & \text{if } \lambda_n(\eta_1) > \mu_n - \varepsilon \\ \lambda_n(\eta_1), & \text{otherwise.} \end{cases} \tag{50}$$

$$\lambda_{\text{out}} = \begin{cases} 0, & \text{if } \lambda_{\text{out}}(\eta_1) < 0 \\ \mu_{\text{core}} - \varepsilon, & \text{if } \lambda_{\text{out}}(\eta_1) > \mu_{\text{core}} - \varepsilon \\ \lambda_{\text{out}}(\eta_1), & \text{otherwise.} \end{cases} \tag{51}$$

$$\lambda_{\text{cloud}} = \begin{cases} 0, & \text{if } \lambda_{\text{cloud}}(\eta_1) < 0 \\ \mu_c - \varepsilon, & \text{if } \lambda_{\text{cloud}}(\eta_1) > \mu_c - \varepsilon \\ \lambda_{\text{cloud}}(\eta_1), & \text{otherwise.} \end{cases} \tag{52}$$

Substitute (50) into (49) and compute $g_2(\eta_1) = \lambda_{\text{cloud}} - \lambda_{\text{out}}$ according to (52) and (51), then we can represent g_2 as the function of η_1 within the inequation constraints of **P2**.
Step 3. Search for η_1 that enforces $g_2(\hat{s}(\eta_1)) = 0$. We can prove that $g_2(\eta_1)$ increases with η_1. Thus, the η_1 satisfying $g_2(\eta_1) = 0$ can be obtain by the bisection method.

Proof. Substitute (6) and (47) into (44), then $g_2(\eta_1)$ can be given as

$$\begin{aligned} g_2(\eta_1) &= \lambda_{\text{cloud}}(\eta_1) - \lambda_{\text{out}}(\eta_1) \\ &= (\mu_c - \sqrt{\frac{-V\mu_c}{\eta_2(\eta_1)A}}) - (A - \sum_{n=1}^{N}\lambda_n) \\ &= \mu_c - A - \sqrt{\frac{-V\mu_c}{\eta_2(\eta_1)A}} + \sum_{n=1}^{N}\lambda_n, \end{aligned} \tag{53}$$

where $\eta_2(\eta_1)$ and λ_n are presented in (49) and (50). It can be derived that λ_n increases with η_1, and $\eta_2(\eta_1)$ decreases with η_1. Therefore, $g_2(\eta_1)$ increases with η_1. □

With above steps, we have obtained an efficient algorithm of problem **P2** (which is also proved to be effective in Sec. 6). Thus we can solve the original problem **P1** as follows: 1) In each time slot t, update $m(t)$ according to (11) and compute $Q(t)$ according to (18). 2) Traverse $x(t) \in \mathbb{L}$ and solve problem **P2** with the above three steps. We summarize the details in Algorithm 1.

Algorithm 1 WiDaS Algorithm

```
1:  for each $t \in \{1, 2, ..., T\}$ do
2:      Update $m(t)$ according to (11).
3:      Update $Q(t)$ according to (18).
4:      for each $x(t) \in \{0, 1, ..., L\}$ do
5:          $\mu_c(t) = m(t)\mu_{\text{ins}}$.
6:          $\mu_{\text{core}}(t) = R(x(t))$.
7:          $C(t) = m(t)p_{\text{ins}} + P(x(t))$.
8:          NUM $= \left\lceil \log_2(\frac{\eta_1^R - \eta_1^L}{\sigma}) \right\rceil - 1$.
9:          $\eta_1^m = \frac{\eta_1^L + \eta_1^R}{2}$.
10:         for num $= 0$ : NUM do
11:             Compute $g_2(\hat{s}(\eta_1^L))$ and $g_2(\hat{s}(\eta_1^m))$ according to (51), (52).
12:             if $g_2(\hat{s}(\eta_1^m)) = 0$ then
13:                 break.
14:             else if $g_2(\hat{s}(\eta_1^1)) \cdot g_2(\hat{s}(\eta_1^m)) < 0$ then
15:                 $\eta_1^R = \eta_1^m$.
16:             else
17:                 $\eta_1^L = \eta_1^m$.
18:             end if
19:             $\eta_1^m = \frac{\eta_1^L + \eta_1^R}{2}$.
20:         end for
21:         $\eta_1 = \eta_1^m$.
22:         Compute $\hat{s} = \langle \lambda_1, ..., \lambda_N, \lambda_{\text{out}}, \lambda_{\text{cloud}} \rangle$ according to (50), (51), (52).
23:         Compute $Q(t)(C(t) - \Theta) + VD(t)$.
24:     end for
25:     $x^*(t) = \underset{x(t)\in\{0,1,...l\}}{\arg\min} \{Q(t)(C(t) - \Theta) + VD(t)\}$.
26:     $\rho(x^*(t)) = \frac{\lambda_{\text{cloud}}(x^*(t))}{\mu_c(t)}$.
27: end for
```

The main computation of Algorithm 1 comes from Line 11. For each η_1, computing $g_2(\hat{s}(\eta_1))$ requires traversing λ_n ($n \in \mathbb{N}$) to restrict within the equation constraints, causing the complexity of $O(N)$. For each $x(t)$, searching for the optimal η_1 with the bisection method yields $O(\log(\frac{\eta_1^R - \eta_1^L}{\sigma}))$ iterations. Therefore, the total computing complexity of Algorithm 1 is $O(TLN\log(\frac{\eta_1^R - \eta_1^L}{\sigma}))$, where η_1^L, η_1^R are the left and right bound of the bisection method, and σ is the searching precision.

Remark: Although the water-filling method has been widely adopted in the wireless communication area, we are the first to introduce the water-filling method into workload scheduling (considering both task transmission and task processing) in mobile edge computing. Apart from the lower bound (i.e., 0, which is the same as the above cases in wireless communications), the workload scheduling results in mobile edge computing also have upper bounds (related to the edge resource capacities), which further aggravates the difficulty of problem analysis and addressing. Existing water-filling based solutions have demonstrated that the waterfilling-like optimization problems with a single water level can be solved by iterative algorithms and exact algorithms [46]. In this paper, we first derive the workload scheduling results into a waterfilling-like form with a single water level (in Eq. (43) and (44)). As the iterative algorithms for waterfilling-like optimization problems with a single water level can get close to the exact value when the number of iterations goes to infinity [20], our proposed algorithm can get to the *optimal solution (same as the KKT-based algorithm) when the searching precision σ go to zero.*

5.3 Discussion on Decentralized Implementation

Algorithm 1 is designed under the ETSI MEC architecture, in which all the edge nodes belong to the same administrator or operator (e.g., a mobile network operator or a cloud service provider), thus all of them can be managed in a centralized manner. However, there are also scenarios that the edge nodes consist of multiple entities pursuing their own interests. In these scenarios, not all edge nodes are willing to share their spare resources cooperatively and centralized control over the whole system is not feasible. Decentralized implementation of the proposed algorithm should be taken into consideration. Coalition games can be utilized to characterize the behaviours of edge nodes. Edge nodes are motivated to form small coalitions by proper incentive mechanisms. With these incentive mechanisms, all the edge nodes within one coalition can increase their utility by working cooperatively while edge nodes across different edge nodes prefer to be segregated. In this case, our proposed algorithm can be applied to scheduling mobile tasks among the edge nodes that are within one coalition. It is not hard to notice that proper incentive mechanisms are the key to the coalition games. Based on the analysis in [47], [48], merge-and-split operations can be used to form proper coalitions and the stability can be enforced.

6 Performance Evaluation

In this section, we conduct extensive simulations to evaluate WiDaS under different task arrival patterns.

6.1 Simulation Setup and Metrics

We provide a simulator to realize the functionality of MEO which manages the MEC application instantiation and traffic scheduling over the cloud-assisted mobile edge computing system . We consider a system with $N = 10$ edge nodes,

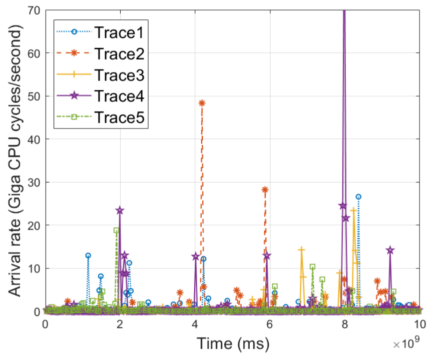


Fig. 3. Tracelogs of Google compute cells.

which have uniformly distributed computation capacities in [10,20] Giga CPU cycles/second. The required CPU cycles of mobile tasks follow exponential distribution with the expectation of $\xi = 1$ Giga CPU cycles. The task arrival process at each edge node is a non-homogeneous Poisson process with the expected arrival rate $A_n(t)$ uniformly distributed in [5,15] tasks/second. Nearby edge nodes are connected by LAN or wired P2P connection (with migration delay of $d_n = 10$ ms) and each edge node can connect to the cloud though the core network. We use the Amazon on-demand instance, m4.large [49], to process excessive tasks from edge nodes, with the computation capacity of 7.8 Giga CPU cycles/second and the pricing rate of 0.1 dollar/hour per instance. When migrating mobile tasks to nearby edge nodes or outsourcing these tasks to the cloud, input data of these tasks should be transmitted on the LAN or in the core network. Take augmented reality tasks as an example. When migrating augmented reality tasks among edge nodes or to the cloud, slotted frames of videos, i.e., figures, should be transmitted. As processing one 420 kB figure requires around 1 Giga CPU cycles computation [50], we set the transmission ratio $c = 3.36$ Mbits/Giga CPU cycles.

6.1.1 Task Arrival Pattern

We use three types of traffic patterns to simulate the time-varying property of mobile task arrivals at edge nodes, including one trace-driven traffic pattern and two mathematical traffic patterns.

Trace-driven traffic pattern: Mobile edge computing has not been widely deployed in practice, thus we can not trace the mobile task arrivals at edge nodes precisely. In this work, we use the task arrival traces of Google compute cells to simulate the mobile task arrivals at edge nodes. In Google clusters, a compute cell is composed of several well connected computing machines with high-bandwidth network, which is similar to an edge node. We employ the tracelogs of Google compute cells, i.e., Google cluster usage traces [51], to imitate the real traces of mobile task arrivals at edge nodes. These traces record the timestamps of task events (e.g., submit, schedule, finish, etc.) in task event tables and the CPU usage of compute cells in resource usage tables. The task arrival rates in terms of computation requests are computed as

$$A_{\text{cell}} = (t^{\text{finish}} - t^{\text{schedule}}) \cdot U^{\text{cpu}} \cdot \mu^{\text{cpu}} \cdot r^{\text{task}}, \tag{54}$$

where t^{finish} and t^{schedule} are the timestamps of task finishing and scheduling to machines in a compute cell. U^{cpu} represents the average CPU usage and μ^{cpu} is the serving rate of a CPU. r^{task} is the number of tasks arriving at the cell within a time slot. We obtain 5 traces of task arrivals over a 10^{10} ms period by slightly modifying the traces of a Google compute cell (similar to [7]). The results are shown in Fig. 3.

Random traffic pattern: Each edge node has mobile tasks with random arrival rates uniformly distributed in [5,15] tasks/second.

Normal traffic pattern: Each edge node has mobile tasks with normally distributed arrival rates through T time slots with the expectation 10 tasks/second, and standard deviation 2 tasks/second. Note that we take the absolute value if the arrival rate at any time slot is negative.

6.1.2 Benchmark Algorithms

We compare WiDaS with three benchmark algorithms:

KKT-based algorithm: The KKT-based algorithm computes the optimal solution by directly solving the convex optimization problem **P2** with KKT conditions in each time slot. According to the analysis in Sec. 5.1, the computation complexity of the KKT-based algorithm is $O(TL \cdot 4^N)$.

Edge-first algorithm: Mobile tasks are processed at edge nodes with high priority if the edge nodes have sufficient computation capabilities (i.e., $\mu_n > A_n(t)$). Otherwise, the excessive tasks are outsourced to the cloud.

Fair-ratio algorithm: The mobile tasks scheduled to an edge node are proportional to the task arrival at the edge node, with the ratio identical among all the edge nodes.

6.1.3 Metrics

In the simulation results, we take the average response time in T time slots, i.e., $\frac{1}{T}\sum_{t=0}^{T-1} D(t)$, as the metrics of system performance, and the average system cost in T time slots, i.e., $\frac{1}{T}\sum_{t=0}^{T-1} C(t)$, as the metrics of system cost.

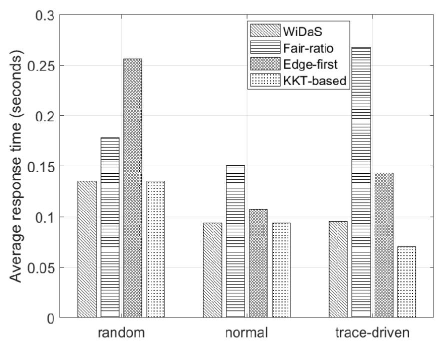
6.2 Efficiency and Effectiveness Evaluation

We compare WiDaS with the benchmark algorithms by conducting simulations under the three traffic patterns. The results demonstrate that WiDaS shows two-fold benefits of high efficiency and effectiveness. We first illustrate the average results of different algorithms in Fig.4. To further illustrate the details of time-varying response time and system cost, we present the results over 100 time slots in Fig. 5.

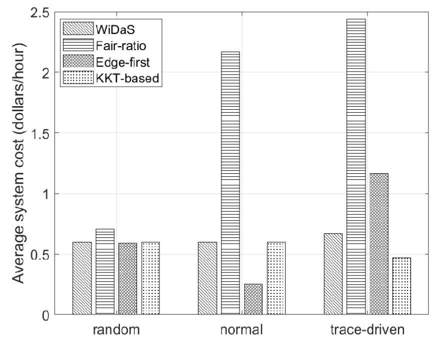
6.2.1 Efficiency

The simulation results show that WiDaS can achieve the approximate results with the KKT-based algorithm while reducing the execution time significantly.

As shown in Fig. 4 and Fig. 5, in terms of both response time and system cost, WiDaS has the approximate results with the KKT-based algorithm. In the simulation under the trace-driven traffic pattern, WiDaS has slightly higher response time and system cost than the KKT-based algorithm. This is because under the trace-driven task arrival

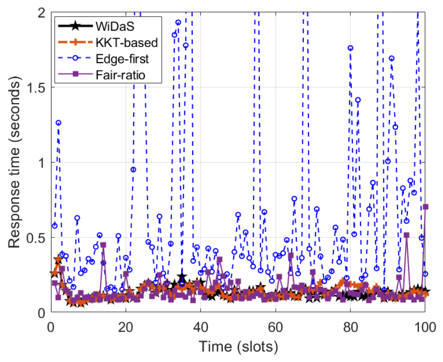


(a) Response time

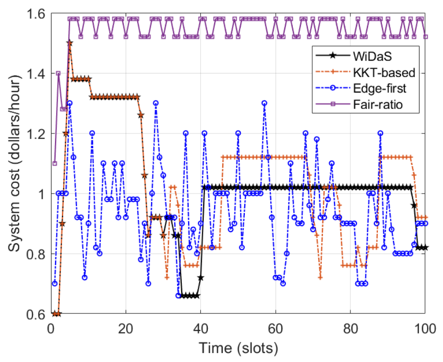


(b) System cost

Fig. 4. Results of algorithms under different task arrival patterns.



(a) Response time



(b) System cost

Fig. 5. The response time and system cost in different time slots: random task arrival pattern, $V = 5$, $\theta = 1$ dollar/h.

pattern, mobile task arrivals at edge nodes are much more fluctuated with time. Removing the heterogenous computation capacity constraints (i.e., the inequation constraints in **P2**) of edge nodes results in performance degrade in the WiDaS algorithm. According to the complexity analysis in Sec. 5.1 and 5.2, the KKT-based algorithm has an exponential complexity while WiDaS has a polynomial complexity. We have also tested the execution time by repeating for 100 times, and the WiDaS algorithm can complete in 9.2ms in each time slot while the KKT-based algorithm requires 902.5ms (Surface laptop 2018 version, 13.5-inch, MATLAB R2017b). Therefore, WiDaS has the benefit of high efficiency.

6.2.2 Effectiveness

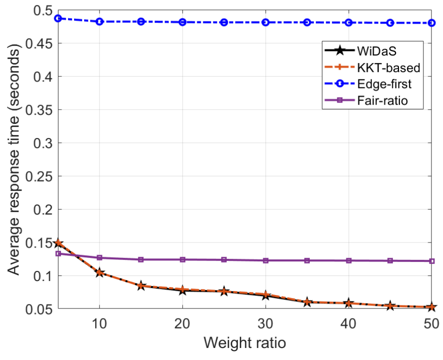
The simulation results in Fig. 4(a) demonstrate that WiDaS can reduce the average response time by 24.1%-64.4% over the Fair-ratio algorithm and 12.7%-47.2% over Edge-first algorithm with a reduced system cost.

In terms of response time, compared with the Fair-ratio algorithm, WiDaS reduces the average response time by 24.1%, 37.7%, and 64.4% under the random, normal, and trace-driven task arrival pattern, respectively. In the WiDaS algorithm, the heterogeneity of edge computation capacities is taken into consideration and the loads of edge nodes can be well balanced. Thus, the average response time can be effectively reduced over the Fair-ratio algorithm. Compared with the Edge-first algorithm, WiDaS reduces the average response time by 47.2%, 12.7%, and 33.4% under the three task arrival patterns. In the Edge-first algorithm, the computation delay at each edge node is highly dependent on the task arrival rate in each time slot, thus the response time fluctuates greatly with time, as shown in Fig. 5(b). The WiDaS algorithm can take advantage of cloud resource agility to accommodate the dynamic tasks at edge nodes and can fully utilize the system resources through task migration among unbalanced edge nodes. Therefore, WiDaS can effectively reduce the response time over the Edge-first algorithm.

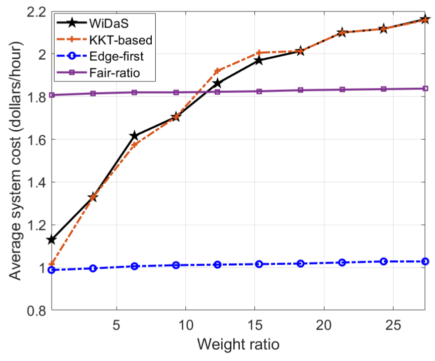
In terms of system cost, WiDaS can maintain the average system cost under the budget and can effectively reduce the average system cost over the Fair-ratio algorithm, as shown in Fig.4(b). The Fair-ratio algorithm does not consider the heterogeneity of edge computation capacities, and thus can not fully utilize the computation capacities of edge nodes, leading to high outsourcing cost. The Edge-fist algorithm processes mobile tasks at edge nodes with higher priority and only uses cloud resources to process the excessive tasks from edge nodes. Thus, the usage of cloud resources is highly dependent on the arrival rates of mobile tasks, and the system cost of the Edge-first algorithm varies greatly with time, as shown in Fig. 5(b). When edge nodes can provide sufficient computation capacities for mobile tasks, the Edge-first algorithm has lower system cost, e.g., under the normal traffic pattern in Fig. 4(b). Otherwise, the Edge-first algorithm has higher system cost than the WiDaS algorithm, such as under the trace-driven traffic pattern in Fig. 4(b).

6.3 Performance-cost Tradeoff

We experimentally show the relationship between the average response time and the system cost, helping system



(a) Response time with V



(b) System cost with V

Fig. 6. Performance-cost tradeoff with V, $\theta = 2$ dollar/h.

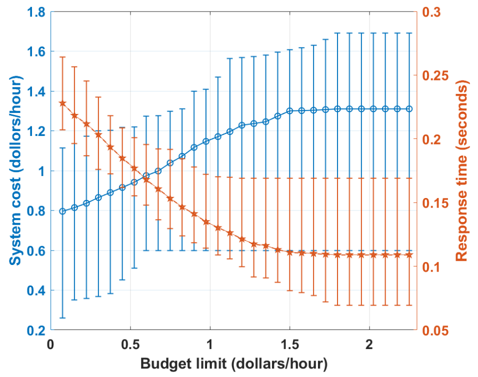


Fig. 7. The average results of performance-cost tradeoff with 95% confidence interval by changing θ, $V = 5$.

administrators to properly trade off the system performance and cost. According to the conclusion of Theorem 1, the performance-cost tradeoff relies on the weight ratio V. In this section, we first illustrate the average response time and system cost with changing V in Fig. 6. Then we show the relationship between the response time and system cost by changing the budget limit in Fig. 7.

Our simulation results show that the average system cost of WiDaS increases with the weight ratio V while the response time decreases with V. In the Edge-first and the Fair-ratio algorithm, the usage of cloud resources and task scheduling results mainly rely on mobile task arrivals and edge computation capabilities, thus the system cost and the response time do not change significantly with V. When V is small, the WiDaS algorithm decreases the usage of cloud resources to reduce the system cost. Most mobile tasks are processed at edge nodes, incurring high response time. As V increases, the response time becomes more dominant in the objective. To reduce the response time, WiDaS outsources more tasks to the cloud, resulting in high system cost.

The simulation results demonstrate that the average response time of WiDaS decreases with the budget limit θ while the average system cost increases with θ. Fig. 7 shows the average results of performance-cost tradeoff with 95% confidence interval. Note that as the we obtain the results with the Monte Carlo method [52], i.e., simulations are repeated with the edge task arrivals and computation capacities uniformly distributed among certain value ranges in each simulation, the ranges of the results are directly influenced by the value ranges of the edge computation capacities and task arrivals.

In these simulations, the value range of task arrivals and computation capacities is large, thus, the results with 95% confidence interval are also significantly scattered. It can be observed that when the budget limit is significantly high (larger than 1.5 dollars/hour), the average system cost no longer increases and the average response time can not be further reduced. This is because when the budget limit is high, sufficient cloud computation resources can be tenanted. The communication delay among edge nodes or from edge nodes to the cloud becomes the bottleneck to reduce the average response time. Tenanting more cloud instances can not further reduce the response time.

7 Conclusions

In this paper, we have solved dynamic task scheduling in the cloud assisted mobile edge computing system to facilitate the functionality of the MEO in the ETSI MEC architecture. The problem has been formulated as a dynamic optimization problem with queuing analysis, aiming at optimizing the average task response time within the resource budget limit. We have proposed the WiDaS algorithm to solve this problem, which is proved to have polynomial computation complexity. Extensive simulations have been conducted to evaluate WiDaS under different task arrival patterns, and the results demonstrate that WiDaS shows two-fold benefits of high efficiency and effectiveness. For the future work, we will investigate the workload scheduling problem in the cloud-edge-device system, in which the capabilities of mobile devices are also taken into consideration.

References

[1] B. G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proc. ACM European Conference on Computer Systems (EuroSys'11)*, Salzburg, Austria, Apr. 2011, pp. 301–314.

[2] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *Proc. ACM International Conference on Mobile Systems, Applications, and Services (MobiSys'10)*, San Francisco, California, USA, June 2010, pp. 49–62.

[3] R. K. Balan, J. Flinn, M. Satyanarayanan, S. Sinnamohideen, and H. I. Yang, "The case for cyber foraging," in *Proc. ACM the 10th workshop on ACM SIGOPS European workshop*, Saint-Emilion, France, Jul. 2002, pp. 87–92.

[4] "Mobile edge computing (mec); framework and reference architecture," White Paper, ETSI GS MEC 003 V1.1.1, Mar. 2016.

[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, Oct. 2016.

[6] F. Boccardi, R. W. Heath, A. Lozano, T. L. Marzetta, and P. Popovski, "Five disruptive technology directions for 5g," *IEEE Communications Magazine*, vol. 52, no. 2, pp. 74–80, Feb. 2014.

[7] X. Ma, S. Wang, S. Zhang, P. Yang, C. Lin, and X. S. Shen, "Cost-efficient resource provisioning for dynamic requests in cloud assisted mobile edge computing," *IEEE Transactions on Cloud Computing*, 2019.

[8] L. Chen, S. Zhou, and J. Xu, "Computation peer offloading for energy-constrained mobile edge computing in small-cell networks," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1619–1632, 2018.

[9] "Developing software for multi-access edge computing," White Paper, ETSI, Feb. 2019.

[10] M. J. Neely, "Stochastic network optimization with application to communication and queueing systems," *Synthesis Lectures on Communication Networks*, vol. 3, no. 1, pp. 1–211, Sep. 2010.

[11] A. Al-Shuwaili and O. Simeone, "Energy-efficient resource allocation for mobile edge computing-based augmented reality applications," *IEEE Wireless Communications Letters*, vol. 6, no. 3, pp. 398–401, Jun. 2017.

[12] S. Zhang, P. He, K. Suto, P. Yang, L. Zhao, and X. Shen, "Cooperative edge caching in user-centric clustered mobile networks," *IEEE Transactions on Mobile Computing*, vol. 17, no. 8, pp. 1791–1805, 2017.

[13] P. Yang, N. Zhang, S. Zhang, L. Yu, J. Zhang, and X. S. Shen, "Content popularity prediction towards location-aware mobile edge caching," *IEEE Transactions on Multimedia*, 2018.

[14] S. Zhang, W. Quan, J. Li, W. Shi, P. Yang, and X. Shen, "Air-ground integrated vehicular network slicing with content pushing and caching," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 9, pp. 2114–2127, 2018.

[15] S. Sundar and B. Liang, "Offloading dependent tasks with communication delay and deadline constraint," in *IEEE Conference on Computer Communications (INFOCOM'18)*. IEEE, 2018, pp. 37–45.

[16] Y. Wen, W. Zhang, and H. Luo, "Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones," in *Proc. IEEE International Conference on Computer Communications (INFOCOM'12)*, Orlando, Florida, USA, May 2012, pp. 2716–2720.

[17] X. Chen, "Decentralized computation offloading game for mobile cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 4, pp. 974–983, 2015.

[18] C. Xian, Y. Lu, and Z. Li, "Adaptive computation offloading for energy conservation on battery-powered systems," in *Proc. IEEE International Conference on Parallel and Distributed Systems (ICPADS'07)*, Hsinchu, Taiwan, Dec. 2007, pp. 1–8.

[19] J. Zheng, Y. Cai, Y. Wu, and X. Shen, "Dynamic computation offloading for mobile cloud computing: A stochastic game-theoretic approach," *IEEE Transactions on Mobile Computing*, vol. 18, no. 4, pp. 771–786, 2018.

[20] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, Oct. 2016.

[21] S. Guo, J. Liu, Y. Yang, B. Xiao, and Z. Li, "Energy-efficient dynamic computation offloading and cooperative task scheduling in mobile cloud computing," *IEEE Transactions on Mobile Computing*, vol. 18, no. 2, pp. 319–333, 2018.

[22] M. V. Barbera, S. Kosta, A. Mei, and J. Stefa, "To offload or not to offload? the bandwidth and energy costs of mobile cloud computing," in *Proc. IEEE International Conference on Computer Communications (INFOCOM'13)*, 2013, pp. 1285–1293.

[23] M. R. Rahimi, N. Venkatasubramanian, and A. V. Vasilakos, "Music: Mobility-aware optimal service allocation in mobile cloud computing," in *Proc. IEEE International Conference on Cloud Computing (CLOUD'13)*, Santa Clara, CA, USA, 2013, pp. 75–82.

[24] Y. Shi, S. Chen, and X. Xu, "Maga: A mobility-aware computation offloading decision for distributed mobile cloud computing," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 164–174, 2017.

[25] 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.

[26] L. Yang, J. Cao, Z. Wang, and W. Wu, "Network aware mobile edge computation partitioning in multi-user environments," *IEEE Transactions on Services Computing*, 2018.

[27] H. Li, K. Ota, and M. Dong, "Learning iot in edge: Deep learning for the internet of things with edge computing," *IEEE network*, vol. 32, no. 1, pp. 96–101, 2018.

[28] J. Kwak, Y. Kim, J. Lee, and S. Chong, "Dream: Dynamic resource and task allocation for energy minimization in mobile cloud systems," *IEEE Journal on Selected Areas in Communications*, vol. 33, no. 12, pp. 2510–2523, 2015.

[29] X. Ma, S. Zhang, W. Li, P. Zhang, C. Lin, and X. Shen, "Cost-efficient workload scheduling in cloud assisted mobile edge computing," in *Proc. IEEE/ACM International Symposium on Quality of Service (IWQoS'17)*, Vilanova i la Geltr, Spain, Jun 2017.

[30] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *IEEE Conference on Computer Communications (INFOCOM'18)*. IEEE, 2018, pp. 207–215.

[31] Y. Cui, J. Song, K. Ren, M. Li, Z. Li, Q. Ren, and Y. Zhang, "Software defined cooperative offloading for mobile cloudlets," *IEEE/ACM Transactions on Networking (TON)*, vol. 25, no. 3, pp. 1746–1760, 2017.

[32] J. P. V. Champati and B. Liang, "Delay and cost optimization in computational offloading systems with unknown task processing times," *IEEE Transactions on Cloud Computing*, 2019.

[33] R. Deng, R. Lu, C. Lai, T. H. Luan, and H. Liang, "Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 1171–1181, 2016.

[34] S. Jošilo and G. Dán, "Decentralized algorithm for randomized task allocation in fog computing systems," *IEEE/ACM Transactions on Networking*, vol. 27, no. 1, pp. 85–97, 2018.

[35] M. Xu, Z. Fu, X. Ma, L. Zhang, Y. Li, F. Qian, K. L. S. Wang, and X. L. J. Yang, "From cloud to edge: A first look at public edge platforms," in *ACM Internet Measurement Conference (IMC'21)*. Accepted, 2021.

[36] Z. Liu, C. Yuan, C. Bash, A. Wierman, and C. Hyser, "Renewable and cooling aware workload management for sustainable data centers," *Acm SIGMETRICS Performance Evaluation Review*, vol. 40, no. 1, pp. 175–186, 2012.

[37] C. N. Le Tan, C. Klein, and E. Elmroth, "Location-aware load prediction in edge data centers," in *International Conference on Fog and Mobile Edge Computing (FMEC)*. IEEE, 2017, pp. 25–31.

[38] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[39] Amazon. Aws auto scaling. [Online]. Available: https://aws.amazon.com/autoscaling/,2018

[40] H. Pishro-Nik, *Introduction to probability, statistics, and random process*. Galway, Ireland: Kappa Research LLC, 2014.

[41] Amazon. Ec2 pricing. [Online]. Available: https://aws.amazon.com/cn/ec2/pricing/,2018

[42] M. Satyanarayanan, P.Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, Oct. 2009.

[43] Amazon. Aws direct connect. [Online]. Available: https://aws.amazon.com/cn/directconnect/,2018

[44] C. Nguyen, C. Klein, and E. Elmroth, "Elasticity control for latency-intolerant mobile edge applications," in *IEEE/ACM Symposium on Edge Computing (SEC'20)*, 2020, pp. 70–83.

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

[46] D. P. Palomar and J. R. Fonollosa, "Practical algorithms for a family of waterfilling solutions," *Signal Processing IEEE Transactions on*, vol. 53, no. 2, pp. 686–695, 2005.

[47] K. R. Apt and A. Witzel, "A generic approach to coalition formation," *International game theory review*, vol. 11, no. 03, pp. 347–367, 2009.

[48] L. Chen, C. Shen, P. Zhou, and J. Xu, "Collaborative service placement for edge computing in dense small cell networks," *IEEE Transactions on Mobile Computing*, vol. 20, no. 2, 2021.

[49] Amazon-EC2. Ec2 on-demand instance. [Online]. Available: https://amazonaws-china.com/ec2/pricing/,2018

[50] T. Soyata, R. Muraleedharan, C. Funai, M. Kwon, and W. Heinzelman, "Cloud-vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture," in *Proc. IEEE Symposium on Computers and Communications (ISCC'12)*, Cappadocia, Turkey, Jul. 2012, pp. 59–66.

[51] C. Reiss, "Google cluster-usage traces: Format + schema," 2011.
[52] N. Metropolis and S. Ulam, "The monte carlo method," *Journal of the American statistical association*, vol. 44, no. 247, pp. 335–341, 1949.

Xiao Ma received her Ph.D. degree in Department of Computer Science and Technology from Tsinghua University, Beijing, China, in 2018. She is currently a lecturer at the State Key Laboratory of Networking and Switching Technology, BUPT. From October 2016 to April 2017, she visited the Department of Electrical and Computer Engineering, University of Waterloo, Canada. Her research interests include mobile cloud computing and mobile edge computing.

Ao Zhou received the P.H.D degrees in Beijing University of Posts and Telecommunications, Beijing, China, in 2015. She is currently an Associate Professor with State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. She has published 20+ research papers. She played a key role at many international conferences. Her research interests include Cloud Computing and Edge Computing.

Shan Zhang (S'13-M'16) received her Ph.D. degree in Department of Electronic Engineering from Tsinghua University and B.S. degree in Department of Information from Beijing Institute Technology, Beijing, China, in 2016 and 2011, respectively. She is currently with the School of Computer Science and Engineering, Beihang University, Beijing, China. From August 2016 to December 2017, she was a post doctoral fellow in Department of Electronical and Computer Engineering, University of Waterloo, Ontario, Canada. Her research interests include network resource and traffic management, network virtualization and integration. Dr. Zhang received the Best Paper Award at the Asia-Pacific Conference on Communication in 2013.

Qing Li received the B.S. and M.S. degrees from Hebei University in 2014 and Xidian University in 2017, respectively, both in communication engineering. She is currently a Ph.D. candidate at the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. Her research interests include cloud computing and mobile edge computing.

Alex X. Liu received his Ph.D. degree in Computer Science from The University of Texas at Austin in 2006, and is a professor at the Department of Computer Science and Engineering, Michigan State University. He received the IEEE & IFIP William C. Carter Award in 2004, a National Science Foundation CAREER award in 2009, and the Michigan State University Withrow Distinguished Scholar Award in 2011. He has served as an Editor for IEEE/ACM Transactions on Networking, and he is currently an Associate Editor for IEEE Transactions on Dependable and Secure Computing and IEEE Transactions on Mobile Computing, and an Area Editor for Computer Communications. He has served as the TPC Co-Chair for ICNP 2014 and IFIP Networking 2019. He received Best Paper Awards from ICNP-2012, SRDS-2012, and LISA-2010. His research interests focus on networking and security. He is a Fellow of the IEEE.

Shangguang Wang is a Professor at the School of Computing, Beijing University of Posts and Telecommunications, China. He received his Ph.D degreeat Beijing University of Posts and Telecommunications in 2011. He is a Vice-Director of the State Key Laboratory of Networking and Switching Technology. He has published more than 150 papers, and his research interests include service computing, mobile edge computing, cloud computing, and ?Space-air-ground Computing. He served as General Chairs or TPC Chairs of IEEE EDGE 2020, IEEE CLOUD 2020, IEEE SAGC 2020, IEEE EDGE 2018, and IEEE ICFCE 2017, etc., and Vice-Chair of IEEE Technical Committee on Services Computing (2015-2018). He is currently serving as Executive Vice-Chair of IEEE Technical Committee on Services Computing (2021-), and Vice-Chair of IEEE Technical Committee on Cloud Computing (2020-). He is a senior member of the IEEE.