Dynamic User Allocation in Stochastic Edge Computing Systems

Phu Lai, Qiang He, Xiaoyu Xia, Feifei Chen, Mohamed Abdelrazek, John Grundy, John Hosking, and Yun Yang

Abstract—Edge computing (EC) is a new distributed computing paradigm where edge servers are deployed at, or near cellular base stations in close proximity to end-users. This offers computing resources at the edge of the network, facilitating a highly accessible platform for real-time, latency-sensitive services. A typical EC environment is highly stochastic with random user arrivals and departures over time. In this paper, we address the user allocation problem from a service provider's perspective, who needs to allocate its users to the cloud or edge servers in a specific area. A user, who has a multi-dimensional resource requirement, can be allocated to either the remote cloud, which incurs a high latency, or an edge server, which results in a low latency but might require the user to wait in a queue. This paper aims to achieve a controllable trade-off between performance (throughput) and several associated costs such as queuing delay and latency costs. We model this problem as a stochastic optimization problem, propose SUAC (Stochastic User AlloCation) – an online Lyapunov optimization-based algorithm, and prove its performance bounds. The experimental results demonstrate that SUAC outperforms existing approaches, effectively allocating users with a desired trade-off while keeping the system strongly stable.

Index Terms—Edge computing, user allocation, Lyapunov optimization, resource allocation

1 Introduction

In recent years, edge computing (EC) has been introduced to tackle a major challenge in cloud computing – unpredictable and high latency, which is holding back the development of latency-sensitive applications and services such as VR/AR, smart cities, critical system warning, healthcare and so on. In an EC environment, numerous edge servers are distributed at, or near cellular base stations or access points [1], which are much closer to end-users compared to remote cloud servers. Thus, this new distributed computing paradigm remarkably reduces end-to-end latency. A service provider such as Uber or YouTube can deploy its services on edge servers to better serve its users [2]. To minimize non-service areas, i.e., the areas that are not covered by any edge server, the coverage areas of adjacent edge servers usually partially overlap [3]. Fig. 1 depicts an example of an EC system in a small area.

In an edge computing environment, the edge user allocation (EUA) problem has arisen as a critical problem that challenges service providers [4], [5], [6], [7]. As thin clients, e.g., mobile or IoT devices, are not capable of executing computationally demanding tasks, they will be allocated to cloud servers or edge servers, which are more powerful and able to process those tasks. A service provider needs to decide where to allocate its users so that some optimization objectives are achieved. In an EUA scenario, each user consumes a specific amount of computing resources, e.g., CPU, RAM, storage, and bandwidth, on an edge server or a cloud server once allocated. An edge server will not be able to serve a user if it does not meet that user's resource requirement. In this paper, we focus on non-preemptive allocation [8], [9] – allocation without interrupting ongoing services of allocated users, i.e., it is not possible to reallocate allocated users.

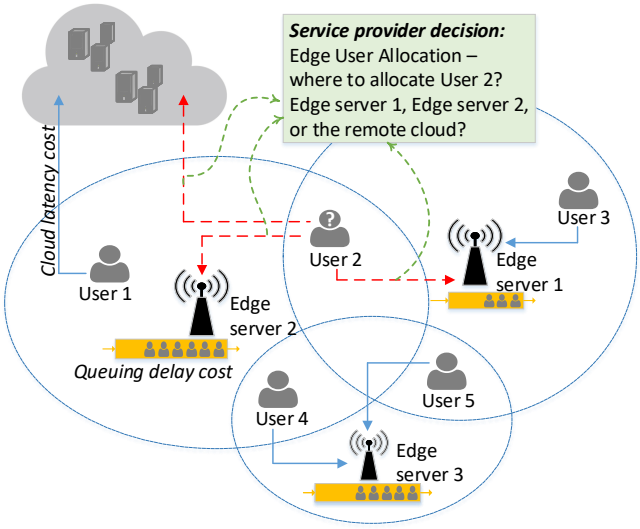


Fig. 1: An example EC system

Ideally, all end-users in a specific geographic area should be allocated to edge servers in their proximity. However, a service provider can only hire a relatively small amount of computing resources since an edge server usually has a very limited computing capacity [10], [11]. It is thus not always

- *P. Lai, Q. He, and Y. Yang are with the School of Software and Electrical Engineering, Swinburne University of Technology, 3122, Australia. E-mail: tlai, qhe, yyang@swin.edu.au.*
- *X. Xia, F. Chen and M. Abdelrazek are with the School of Information Technology, Deakin University, 3125, Australia. E-mail: xiaoyu.xia, feifei.chen, mohamed.abdelrazek@deakin.edu.au.*
- *J. Grundy is with the School of Information Technology, Monash University, 3168, Australia. E-mail: john.grundy@monash.edu.*
- *J. Hosking is with the School of Science, University of Auckland, Auckland, New Zealand. E-mail: j.hosking@auckland.ac.nz.*

possible to serve all users simultaneously. As a consequence, some users may have to wait to be served in a queue. Since an excessively long queue results in a long waiting time, some users will be allocated to the remote cloud to be served by a server in the cloud. As the cloud platform has the elasticity to scale up or down in real-time, it has virtually unlimited computing resources, hence no queuing needed for the users allocated to the remote cloud. This reduces the waiting time to almost zero but at the expense of high latency between the cloud and the users.

Existing research on the EUA problem has a number of limitations, which we address in this paper. 1) *They assume that all the users have identical computing resource requirements or the arrivals of users are always known* [4], [5], [6], [7]. This is unrealistic because the EC environment is highly stochastic. Users with various resource requirements join the EC system dynamically and randomly over time. A user's exact duration in the EC system is also unknown, i.e., they randomly depart from the EC system. One can only give a rough estimate of how long a user would stay in the system. After a user's departure, the computing resources are released to serve new users. 2) *Existing approaches do not consider the fact that users can come and go over time* [4], [6]. Thus proposed approaches can only be executed to allocate users that join the EC system in a single time slot. In the next time slot, when new users arrive, those approaches would have to be re-executed without taking any long-term objectives into account, thus the allocation solutions computed in different time slots might be conflicting each other. 3) More importantly, the limited computing resources in edge computing [10], [11] and the massive number of users in 5G networks [12] further complicate the EUA problem. When the number of users is very large and the EC system does not have sufficient computing resources, *our approach allows users to wait in a distributed queuing system*. In fact, the queuing method has been wildly employed recently in edge computing [3], [13], [14]. The aforementioned existing approaches for EUA assume that users will be allocated to the cloud *immediately* when edge servers are exhausted of computing resources. However, *their models do not incorporate any costs that might incur when users are allocated to the cloud*. In addition, the allocated users do not stay in the EC system permanently. The computing resources released when they leave the EC system can be utilized to serve new users. This also has not been taken into account by existing approaches.

The *throughput* of a time-slotted EC system is defined as the average number of users allocated to edge servers over time. At the beginning of each time slot, a service provider needs to decide where to allocate its users, the remote cloud or which edge server. Under the unpredictability of user arrivals and departures, the objectives of this stochastic EUA problem are twofold: 1) maximizing the throughput benefit by allocating as many users as possible to edge servers as long as it is beneficial to do so, and 2) minimizing the penalty incurred by the queuing delay and cloud-to-user latency. In order to achieve a controllable trade-off between the performance (throughput) and costs, we employ the Lyapunov optimization framework [15], which can make decisions based on the current state of the system without any future information about user arrivals and departures. Its unique advantage over other online optimization approaches is its ability to optimize the performance of a system in a metric (the system benefit measured collectively by the throughput benefit, queuing delay cost, and latency cost in our study), while stabilizing the system (the length of the queues of users waiting to be allocated to each edge server in our study). It can be used to transform a long-term optimization problem into a series of short-term optimization problems, which are to be solved in each time slot. Doing this over multiple time slots will collectively achieve the long-term objective, i.e., system benefit maximization, while stabilizing all the user queues. The main contributions of this paper include:

- We formally model the stochastic EUA problem that aims to help service providers allocate their users in order to achieve their long-term objectives, i.e., to maximize system throughput, and to minimize users' latency and queuing costs while keeping the stochastic EC system strongly stabilized.
- We present SUAC, an online algorithm based on the Lyapunov optimization framework. Although SUAC does not require future information of user arrivals and departures, it can be theoretically shown that SUAC is able to find near-optimal solutions with an $[O(1/V), O(V)]$ performance-cost trade-off.
- A series of experiments are conducted on a real-world dataset to comprehensively evaluate the performance of SUAC.

The organization of this paper is as follows. Section 2 introduces the EC system model. Section 3 first introduces the throughput benefit and associated costs, then formulates the stochastic EUA problem. Section 4 presents the SUAC algorithm, which is then evaluated in Section 5. Section 6 reviews the related literature. Finally, Section 7 concludes the paper and points out future work.

2 System Model

In this section, we formally model the EC system and its associated key characteristics. An EC system in a specific geographical area consists of a set of edge servers. It is a time-slotted system where multiple users arrive in each time slot and need to be allocated to either an edge server or the cloud. Each edge server maintains a queue to hold users that have been allocated to the server but have not been served yet due to insufficient computing resources. The main notations used in this paper are summarized in Table 1.

2.1 System Description

Edge Servers: The set of S edge servers denoted by $\mathcal{S} = \{1, 2, ..., S\}$. Each edge server $s \in \mathcal{S}$ has a certain amount of different computing resource types $\mathcal{R}$ such as CPU, RAM, storage, or bandwidth. Each edge server is equipped with a limited amount of computing resources. The computing capacity of an edge server s is an $|\mathcal{R}|-$dimensional vector C_s. Each edge server covers a particular geographic area, as illustrated in Fig. 1.

Edge Users: There are K types of users categorized by their computing resource requirements, $k \in \mathcal{K} =$

TABLE 1: Main Notations

Notation	Description
$\mathcal{S}$	the set of S edge servers s. $\mathcal{S} = \{1, 2, ..., S\}$
$\mathcal{R}$	the set of computing resource types, or computing capacity dimensions. $\mathcal{R} = \{CPU, RAM, storage, bandwidth, ...\}$
C_s	computing capacity of edge server s. C_s is an $\|\mathcal{R}\|$-dimensional vector where each dimension is the capacity of a resource type in $\mathcal{R}$
$\mathcal{K}$	the set of K user types k, categorized by their computing resource requirements. $\mathcal{K} = \{1, 2, ..., K\}$
$\mathcal{A}_k(t)$	the set of type-k users u ($k \in \mathcal{K}$) arrive in time slot t. The set of all users arrive in time slot t is $\mathcal{A}(t) = \bigcup_{k\in\mathcal{K}} \mathcal{A}_k(t)$
c_u	computing resource requirement of user u. c_u is an $\|\mathcal{R}\|$-dimensional vector where each dimension is the capacity of a resource type in $\mathcal{R}$
$\mathbf{a}_{u,s}(t)$	allocation decision on whether user u will be allocated to edge server s in time slot t
$\mathbf{b}_u(t)$	allocation decision on whether user u will be allocated to the remote cloud in time slot t
$Q_s(t)$	queue backlog of edge server s, i.e., number of users waiting to be served by edge server s, in time slot t
$D_s(t)$	number of users who leave the queue and start being served by edge server s in time slot t
$\overline{b_r}$	time-average benefit gained from system throughput
$\overline{c_d}$	time-average queuing delay for all users
$\overline{c_h}$	time-average latency of all users allocated to the remote cloud
r_s	time-average throughput of edge server s
w_s	the weight that indicates service provider's priority for the throughput benefit gained from serving users by edge server s
N_s	service rate of edge server s, i.e., the maximum number of users can be simultaneously served by edge server s
ℓ	expected user session length
$n_s(t)$	number of users being served by edge server s in time slot t
h_u	user u's cloud-to-user latency
$\omega_r, \omega_d, \omega_h$	normalizing parameters for throughput benefit, queuing delay cost, and latency cost
V	Lyapunov control parameter

$\{1, 2, ..., K\}$. Let an $|\mathcal{R}|-$dimensional vector c_u denote user u's computing resource requirement.

User Arrivals and Departures: The operational timeline of this EC system is discretely slotted with normalized slot duration $t \in \{0, 1, 2, ...\}$. Each time slot t may range from several milliseconds, seconds, to a few minutes, depending on the application context. Let $\mathcal{A}_k(t)$ denote the set of type-k users that arrive in time slot t, the set of all users that arrive in time slot t is $\mathcal{A}(t) = \bigcup_{k\in\mathcal{K}} \mathcal{A}_k(t)$. Each random variable $|\mathcal{A}_k(t)|$, $\forall k \in \mathcal{K}$, is independent of the current number of users in the system. Without loss of generality, we assume that the number of users that arrive in a time slot is bounded, i.e., $\sum_{k\in\mathcal{K}} |\mathcal{A}_k(t)| \leq A^{max}$, $\forall t$, where A^{max} is the maximum of users of all types arrive in a time slot. The time-average arrival rate is given by $\lambda_k = \mathbb{E}\{|\mathcal{A}_k(t)|\}$. The total time-average user arrival rate λ of the system is thus $\sum_{k\in\mathcal{K}} \lambda_k$. A user's duration in the system (the length of a user session) is unknown at all time, i.e., users depart from the EC system randomly, and is measured by the number of time slots. Our approach does not rely on any prior knowledge of the statistics of user arrivals or departures.

2.2 Allocation Decisions

In each time slot, a number of new users arrive and need to be allocated to either 1) ideally, edge servers, or 2) the remote cloud. Let $\mathbf{a}_{u,s}(t), \mathbf{b}_u(t) \in \{0, 1\}$ be the allocation decision to be made for user u in time slot t. We have $\mathbf{a}_{u,s}(t) = 1$ if user u is to be allocated to edge server s, and $\mathbf{b}_u(t) = 1$ if user u is to be allocated to the remote cloud. Let $\mathbf{a}(t) = \{\mathbf{a}_{u,s}(t), \mathbf{b}_u(t)\}_{u\in\mathcal{A}(t)}$ denote the allocation strategy for all users that arrive in time slot t, which must satisfy the following constraints.

Firstly, each user u can be allocated to only one server, either the remote cloud server or an edge server:

$$\mathbf{b}_u(t) + \sum_{s\in\mathcal{S}} \mathbf{a}_{u,s}(t) = 1, \forall u \in \mathcal{A}(t), \forall t \tag{1}$$

Note that we do not consider users who are not located within the coverage of any edge server. A user u can be allocated to an edge server s only if it is located in that edge server's coverage area cov_s (*proximity constraint*):

$$\mathbf{a}_{u,s}(t) = 1 \text{ if } u \in cov_s, \forall u \in \mathcal{A}(t), \forall s \in \mathcal{S}, \forall t \tag{2}$$

and the accumulated computing resource requirements of the users being served by an edge server must not exceed the capacity of that edge server (*capacity constraint*):

$$\sum_{u\in\mathcal{D}(t)} (\mathbf{a}_{u,s}(t) c_u) \preceq C_s, \forall s \in \mathcal{S}, \forall t \tag{3}$$

where $\mathcal{D}(t)$ is the set of users being served by edge server s.

2.3 Queuing Dynamics and Stability

We introduce a distributed queuing architecture where each edge server $s \in \mathcal{S}$ maintains a local queue. If a user is decided to be allocated to an edge server by a mechanism to be discussed later, it will be buffered in the queue of that edge server until that edge server has sufficient computing resources to serve it. We denote queue backlog $Q_s(t)$ as the queue length (number of waiting users) of edge server s in time slot t. We have the following queuing dynamics:

$$Q_s(t+1) = \big[Q_s(t) - D_s(t)\big]_+ + \mathbf{a}_s(t) \tag{4}$$

where $\mathbf{a}_s(t) = \sum_{u\in\mathcal{A}(t)} \mathbf{a}_{u,s}(t)$ is the number of users decided to be allocated to edge server s in time slot t, and $D_s(t)$ is the number of users that leave the queue and start being served by edge server s. Note that $D_s(t)$ is not the number of users who depart from the system (the user session has ended, or the service is no longer required) in time slot t. A queue $Q_s(t)$ is considered *strongly stable* [15] if the queue backlog is bounded: $\lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{Q_s(t)\} < \infty, \forall s \in \mathcal{S}$. Intuitively, this means that the queue length remains finite and does not blow up to infinite over time. An EC system is strongly stable when all the queues' lengths are bounded. We will later theoretically and experimentally show that our approach can stabilize the EC system.

Remark: We consider a distributed queuing model rather than a centralized one, i.e., all edge servers maintain a single queue, for two main reasons. First, the distributed queuing architecture is more common in data centers since the queue memory operates at a much slower speed [8]. Secondly, as the central queue might be far away from users, it would incur extra communication delay, which is not acceptable in the EC environment.

3 PROBLEM FORMULATION

We now identify several key EC components that are taken into account in our model, including the performance (benefit gained from system throughput), and the costs (incurred by queuing delay and cloud-to-user latency).

3.1 Time-Average Throughput

From the service provider's perspective, the overall system throughput, i.e., the number of users allocated to edge servers, is one of the key performance metrics that reflects the quality of service. We define the time-average throughput r_s of each edge server s as follows:

$$r_s = \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\left\{ \sum_{u\in\mathcal{A}(t)} \mathbf{a}_{u,s}(t) \right\}, \forall s \in \mathcal{S} \tag{5}$$

We have $\sum_{s\in\mathcal{S}} r_s$ as the overall throughput of the EC system, which ideally should be maximized and is obviously subject to the following constraint: $\sum_{s\in\mathcal{S}} r_s \leq \lambda$, i.e., the time-average throughput cannot exceed the time-average user arrival rate. The time-average benefit that a service provider gains from system throughput is measured by:

$$\overline{b_r} = \sum_{s\in\mathcal{S}} (w_s r_s) \tag{6}$$

where w_s is a non-negative weight for the throughput r_s of each edge server $s \in \mathcal{S}$. This weight enables the service provider to adjust the benefit gained from different edge servers' throughput. A high value of w_s indicates that, given the same throughput, the benefit gained from edge server s is more significant than other edge servers with lower w_s. For example, if the prices for the computing resources on edge server s are cheaper than those on other edge servers, the throughput on edge server s is more beneficial or profitable.

3.2 Queuing Delay and Latency Cost

For each user, there are two possible allocation options, either the remote cloud or an edge server. Each option is associated with a type of cost. If allocated to an edge server, the user is placed in a queue waiting for its turn to be served since being allocated to an edge server is more desirable, hence highly demanding. This incurs a *queuing delay cost*. If allocated to the remote cloud, which has an ample volume of computing power to serve many users simultaneously, the user skips the queuing but suffers a high cloud latency throughout its service session. This incurs a *latency cost*.

3.2.1 Queuing Delay Cost

The queuing delay cost for a user depends on the current congestion state of its assigned queue and the computing power of the edge server associated with that queue. For example, a more congested queue would apparently lead to a longer queuing delay. However, one also needs to take into account the computing capacity of the edge server associated with that queue. If the edge server's computing capacity is high, it can serve more users simultaneously, resulting in a higher service rate than those with low computing capacities.

The service rate of an edge server s is N_s/ℓ, where ℓ is the expected user session length, measured by the number of time slots, and N_s is the maximum number of users of all types that can be simultaneously served by edge server s in one time slot. In practice, ℓ can be empirically estimated based on historical data. N_s can be calculated based on the computing capacity C_s of edge server s. For example, say there are three possible types of user resource requirements $< 1, 3, 1, 2 >$, $< 2, 1, 3, 1 >$, and $< 3, 1, 2, 2 >$, and edge server s has a capacity of $< 43, 43, 41, 41 >$. This edge server s has enough capacity to serve 22 users concurrently (e.g., 10 users of type $< 1, 3, 1, 2 >$, 5 users of type $< 2, 1, 3, 1 >$, and 7 users of type $< 3, 1, 2, 2 >$).

Let $n_s(t)$ denote the number of users being served by edge server s in time slot t. Then, the queuing delay of a user u who has just arrived at edge server s can be measured by $\frac{[n_s(t)-N_s+Q_s(t)+1]_+}{N_s/\ell}$, where $Q_s(t)$ is the current queue length of edge server s, i.e., the number of queuing users excluding new user arrivals. The estimated queuing delay of another user who arrived right after user u is thus $\frac{[n_s(t)-N_s+Q_s(t)+2]_+}{N_s/\ell}$. Intuitively, $[n_s(t) - N_s + Q_s(t) + i]_+, \forall i \leq \mathbf{a}_s(t)$ represents the number of users queuing ahead of the new users when the edge server is full. $[n_s(t) - N_s + Q_s(t) + i]_+ = 0$ if the server has sufficient resources to serve new users right away without queuing them. We can see that the queuing delay of users allocated to an edge server is proportional to the current queue length $Q_s(t)$ and the number of users allocated to that edge server $\mathbf{a}_s(t)$. Let $M_s(t) = n_s(t) - N_s + Q_s(t)$, the time-average queuing delay for all users in all edge servers can be defined as follows:

$$\overline{c_d} = \sum_{s\in\mathcal{S}} \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\left\{ \frac{[M_s(t)+1]_+}{N_s/\ell} + ... + \frac{[M_s(t)+\mathbf{a}_s(t)]_+}{N_s/\ell} \right\} \tag{7}$$

3.2.2 Latency Cost

If allocated to the remote cloud, the user suffers from high latency throughout its service session. The latency, or communication delay, is influenced by many factors such as transmission medium, distance, bandwidth, etc. To simplify the model, we let h_u be the cloud-to-user latency for user u. The time-average latency of all users allocated to the remote cloud is:

$$\overline{c_h} = \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\left\{ \sum_{u\in\mathcal{A}(t)} (\mathbf{b}_u(t) h_u) \right\} \tag{8}$$

3.3 Time-Average System Benefit Maximization

In the previous sections, we have modeled the throughput benefit $\overline{b_r}$, the queuing delay cost $\overline{c_d}$, and the latency cost $\overline{c_h}$. Now, we formulate the maximization of the time-average system benefit as the following stochastic optimization problem:

$$(\mathbf{P1}) \quad \max_{\mathbf{a}_{u,s},\mathbf{b}_u} \omega_r \overline{b_r} - \omega_d \overline{c_d} - \omega_h \overline{c_h} \tag{9}$$
$$\text{s.t. } (1), (2), (3)$$

where ω_r, ω_d, and ω_h are non-negative weights acting as normalizing parameters for throughput benefit, queuing delay cost, and latency cost, respectively, since they have

different scales. This also allows service providers to adjust their priority for the throughput benefit and other costs. Solving **P1** in an offline and centralized fashion is intractable. Optimally solving this optimization problem requires complete offline future information such as user arrivals and departures, user locations, and user requirements. In the highly stochastic EC environment, that information are often unpredictable and vary over time. Therefore, we tackle this challenge by introducing an online algorithm based on the Lyapunov optimization framework, which can make decisions in every time slot without future information.

4 Stochastic User Allocation

In this section, we introduce the trade-off between the system benefit and system stability, measured by the queue congestion. We then propose SUAC, an online algorithm based on the Lyapunov optimization framework to solve the stochastic EUA problem.

4.1 Stability-Benefit Trade-off

Let $Q(t) = (Q_1(t), ..., Q_S(t))$ be the queue backlog vector. For each time slot t, we define a quadratic Lyapunov function $L(Q(t))$ as:

$$L(Q(t)) \triangleq \frac{1}{2}\sum_{s\in\mathcal{S}} Q_s(t)^2 \tag{10}$$

This function is a scalar measure of the congestion state of the EC system. Intuitively, $L(Q(t))$ is large when at least one of the edge server's queue $Q_s(t)$ is congested, and $L(Q(t))$ is small when all the queue backlogs are small, representing a stable system. Given the Lyapunov function, we define the *conditional Lyapunov drift* to measure the change in this function between two consecutive time slots:

$$\Delta(Q(t)) \triangleq \mathbb{E}\{L(Q(t+1)) - L(Q(t))|Q(t)\} \tag{11}$$

Our objective is to find a user allocation strategy $\mathbf{a}(t)$ to coordinate the queue congestion state, throughput benefit, and other related costs in every time slot. By incorporating the queue stability into the performance-cost trade-off, we come up with a *drift-minus-benefit* expression as follows:

$$\Delta(Q(t)) - V\,\mathbb{E}\{\omega_r\overline{b_r} - \omega_d\overline{c_d} - \omega_h\overline{c_h}|Q(t)\} \tag{12}$$

where V is a non-negative control parameter to balance the trade-off between the drift $\Delta L(Q(t))$ (queue stability) and the system benefit (throughput benefit minus queuing delay and latency costs). $V = 0$ indicates that the system should be stabilized as much as possible regardless of the throughput benefit, i.e., to allocate all users to the remote cloud. Depending on the situation, a service provider can flexibly change the value of V to adjust the trade-off. For example, they can decrease V to keep the queue backlogs small, avoiding system congestion while maximizing the system benefit as much as possible. Under the Lyapunov optimization framework, the user allocation strategy should be chosen to minimize the supremum bound of the above drift-minus-benefit expression (12).

Lemma 1. *Given any allocation strategies in any time slots, the following bound of the drift-minus-benefit* (12) *holds:*

$$\begin{aligned}
&\Delta(Q(t)) - V\,\mathbb{E}\{\omega_r\overline{b_r} - \omega_d\overline{c_d} - \omega_h\overline{c_h}|Q(t)\}\\
&\le B + \sum_{s\in\mathcal{S}}\mathbb{E}\Big\{\boldsymbol{a}_s(t)^2VK + \boldsymbol{a}_s(t)P_s(t)|Q(t)\Big\}\\
&\quad + \sum_{u\in\mathcal{A}(t)}\mathbb{E}\Big\{V\omega_h h_u\big(1 - \sum_{s\in\mathcal{S}}\boldsymbol{a}_{u,s}(t)\big)|Q(t)\Big\}
\end{aligned} \tag{13}$$

where $B = \frac{1}{2}(\sum_{s\in\mathcal{S}} N_s^2 + {A^{max}}^2)$ *is a constant,* $\boldsymbol{a}_s(t) = \sum_{u\in\mathcal{A}(t)}\boldsymbol{a}_{u,s}(t)$, $M_s(t) = n_s(t) - N_s + Q_s(t)$, $P_s(t) = Q_s(t) + V\big(K(2M_s(t)+1) - \omega_r w_s\big)$, *and* $K = \frac{\omega_d\ell}{2N_s}$.

Proof. Let $\mathbf{a}_s(t) = \sum_{u\in\mathcal{A}(t)}\mathbf{a}_{u,s}(t)$. Since $([a-b]_+ + c)^2 \le a^2 + b^2 + c^2 - 2a(b-c), \forall a, b, c \ge 0$, we can derive:

$$\begin{aligned}
&\Delta(Q(t)) = \mathbb{E}\left\{L(Q(t+1)) - L(Q(t))|Q(t)\right\}\\
&= \mathbb{E}\left\{\frac{1}{2}\sum_{s\in\mathcal{S}}\Big(\big([Q_s(t) - D_s(t)]_+ + \mathbf{a}_s(t)\big)^2 - Q_s(t)^2\Big)|Q(t)\right\}\\
&\le \mathbb{E}\left\{\frac{1}{2}\sum_{s\in\mathcal{S}}\Big(D_s(t)^2 + \mathbf{a}_s(t)^2\right.\\
&\qquad\left. - 2Q_s(t)\big(D_s(t) - \mathbf{a}_s(t)\big)\Big)|Q(t)\right\}
\end{aligned} \tag{14}$$

Since $\sum_{s\in\mathcal{S}}\mathbf{a}_s(t) \le A^{max}$ and $\mathbf{a}_s(t) \ge 0, \forall s \in \mathcal{S}$, we have $\sum_{s\in\mathcal{S}}\mathbf{a}_s(t)^2 \le {A^{max}}^2$. In addition, $D_s(t) \le N_s, \forall s \in \mathcal{S}, \forall t$, thus $\sum_{s\in\mathcal{S}}(D_s(t)^2 + \mathbf{a}_s(t)^2)$ is bounded by $\sum_{s\in\mathcal{S}} N_s^2 + {A^{max}}^2$. Also, $-2Q_s(t)D_s(t) \le 0, \forall s \in \mathcal{S}, \forall t$, therefore,

$$\Delta(Q(t)) \le B + \mathbb{E}\Big\{\sum_{s\in\mathcal{S}}\big(\mathbf{a}_s(t)Q_s(t)\big)|Q(t)\Big\} \tag{15}$$

where $B = \frac{1}{2}(\sum_{s\in\mathcal{S}} N_s^2 + {A^{max}}^2)$ is a constant.

By subtracting $V\,\mathbb{E}\{\omega_r\overline{b_r} - \omega_d\overline{c_d} - \omega_h\overline{c_h}\}$ from both sides of (15), we have:

$$\begin{aligned}
&\Delta(Q(t)) - V\,\mathbb{E}\{\omega_r\overline{b_r} - \omega_d\overline{c_d} - \omega_h\overline{c_h}|Q(t)\}\\
&\le B + \mathbb{E}\Big\{\sum_{s\in\mathcal{S}}\big(\mathbf{a}_s(t)Q_s(t)\big)|Q(t)\Big\} - V\,\mathbb{E}\Big\{\omega_r\sum_{s\in\mathcal{S}}(w_s r_s)\\
&\qquad - \omega_d\sum_{s\in\mathcal{S}}\big(\frac{[M_s(t)+1]_+}{N_s/\ell} + ... + \frac{[M_s(t)+\mathbf{a}_s(t)]_+}{N_s/\ell}\big)\\
&\qquad - \omega_h\sum_{u\in\mathcal{A}(t)}\big(\mathbf{b}_u(t)h_u\big)|Q(t)\Big\}\\
&\overset{\dagger}{\le} B + \mathbb{E}\Big\{\sum_{s\in\mathcal{S}}\big(\mathbf{a}_s(t)Q_s(t)\big)|Q(t)\Big\} - V\,\mathbb{E}\Big\{\sum_{s\in\mathcal{S}}(\omega_r w_s\mathbf{a}_s(t))\\
&\qquad - \sum_{s\in\mathcal{S}}\frac{\omega_d\ell(2M_s(t) + 1 + \mathbf{a}_s(t))\mathbf{a}_s(t)}{2N_s}\\
&\qquad - \omega_h\sum_{u\in\mathcal{A}(t)}\Big(h_u\big(1 - \sum_{s\in\mathcal{S}}\mathbf{a}_{u,s}(t)\big)\Big)|Q(t)\Big\}\\
&= B + \sum_{s\in\mathcal{S}}\mathbb{E}\Big\{\mathbf{a}_s(t)^2VK\\
&\qquad + \mathbf{a}_s(t)\Big(Q_s(t) + V\big(K(2M_s(t)+1) - \omega_r w_s\big)\Big)|Q(t)\Big\}\\
&\quad + \sum_{u\in\mathcal{A}(t)}\mathbb{E}\Big\{V\omega_h h_u\big(1 - \sum_{s\in\mathcal{S}}\mathbf{a}_{u,s}(t)\big)|Q(t)\Big\}
\end{aligned} \tag{16}$$

where $K = \frac{\omega_d \ell}{2N_s}$ and $M_s(t) = n_s(t) - N_s + Q_s(t)$. The inequality † is because of the constraint (1) and the fact that $[a+1]_+ + ... + [a+n]_+ \geq na + n(n+1)/2, \forall a \in \mathbb{R}, \forall n \geq 0$. Therefore, Lemma 1 holds. □

We have now transformed the stochastic optimization problem **P1** into the bounding of the drift-minus-benefit. In the next section, we propose an online allocation algorithm given the drift-minus-benefit bound found in Lemma 1.

Remark: This model is easily extensible by incorporating one or more virtual queues into (10). A virtual queue could be a deficit queue of energy consumption, operating budget, etc., allowing service providers to enforce more constraints.

4.2 Stochastic User Allocation Algorithm

In this section, we propose SUAC, a Stochastic User AlloCation algorithm (Algorithm 1), which observes the state of the EC system in every time slot t and determines an allocation strategy $\mathbf{a}(t)$ to minimize the supreme bound on the drift-minus-benefit (12), i.e., the right-hand side of (13). Employing the concept of opportunistically minimizing an expectation, this can be achieved by solving the optimization problem **P2** below.

$$(\mathbf{P2}) \min_{\mathbf{a}_{u,s}, \mathbf{b}_u} \sum_{s \in \mathcal{S}} \Big(\mathbf{a}_s(t)^2 VK + \mathbf{a}_s(t) P_s(t) \Big) + \sum_{u \in \mathcal{A}(t)} \Big(V \omega_h h_u \big(1 - \sum_{s \in \mathcal{S}} \mathbf{a}_{u,s}(t)\big) \Big) \quad (17)$$
$$\text{s.t. (1), (2), (3)}$$

Algorithm 1 SUAC ALGORITHM

Input: $\mathcal{S}, V, \omega_r, \omega_d, \omega_h$
Output: allocation decisions $\mathbf{a}_{u,s}(t), \mathbf{b}_u(t), \forall t, \forall s \in \mathcal{S}$

1: **for** each time slot $t = 0, 1, ..., \infty$ **do**
2: Observe incoming users $\mathcal{A}(t)$ and edge servers' queues $Q_s(t), \forall s \in \mathcal{S}$;
3: Choose $\mathbf{a}_{u,s}(t), \mathbf{b}_u(t), \forall u \in \mathcal{A}(t)$ by solving **P2**;
4: Update the queue $Q_s, \forall s \in \mathcal{S}$ according to (4);
5: **end for**

In every time slot, new users arrive and then get allocated to either the remote cloud or edge servers depending on the solutions to optimization problem **P2**. In the meantime, each edge server serves the users waiting in its queue if it has sufficient computing resources. After a random number of time slots, those users depart and release the computing resources so that the queuing users can start using the service on a first-come, first-served (FCFS) basis. Note that SUAC requires neither the knowledge of future user arrivals nor general statistical user distributions. Given the pre-defined parameters $V, \omega_r, \omega_d, \omega_h$, SUAC can achieve a controllable trade-off between throughput and other associated costs while guaranteeing a stable EC system. In other words, all the edge servers' queues are stable at all times, bounding users' expected queuing delay. Problem **P2** can be solved using an integer programming solver, e.g., IBM ILOG CPLEX Optimizer[1] or Gurobi[2]). According to our experiments, IBM ILOG CPLEX Optimizer can handle as many as 1,000 users arriving at 26 edge servers in each 30-second time slot.

SUAC is an online algorithm that allocates users as they arrive in the EC system. When a user moves within the same edge server's coverage area across two time slots, they will not be considered as a new user. If they move from one edge server's coverage area into another edge server's coverage area across two time slots, they will be allocated as a new user in the second time slot. In this way, SUAC can accommodate user mobility. Within one time slot, we study the quasi-static scenarios where users do not move across edge servers, similar to many other studies [3], [4], [13], [16].

The following theorem demonstrates the existence of a performance-cost trade-off $[O(1/V), O(V)]$ in the proposed SUAC algorithm, allowing service providers to adjust the control parameter V to achieve its desired trade-off between the throughput benefit and other associated costs, namely queuing delay and latency costs.

Theorem 1. *For any user arrival rate in any time slot, employing SUAC with any non-negative V satisfies the following performance bounds:*

1) The time-average system benefit is within a gap (B/V) to the optimal solution:

$$\lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\left\{ \omega_r \overline{b_r} - \omega_d \overline{c_d} - \omega_h \overline{c_h} \right\} \geq \beta^* - \frac{B}{V} \quad (18)$$

2) The average queue backlog is upper bounded:

$$\lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T-1} \sum_{s \in \mathcal{S}} \mathbb{E}\{Q_s(t)\} \leq B + V(\beta^* - \beta^{min}) \quad (19)$$

*where $\beta^* = \omega_r \overline{b_r^*} - \omega_d \overline{c_d^*} - \omega_h \overline{c_h^*}$, and $\overline{b_r^*}, \overline{c_d^*}$, and $\overline{c_h^*}$ are the optimal values of Problem **P2**, and $B = \frac{1}{2}(\sum_{s \in \mathcal{S}} N_s^2 + A^{max^2})$, and β^{min} is defined in the proof.*

Proof. First, we prove the first part of Theorem 1. Using the result obtained in Theorem 4.5 in [15], we can show that there exists a stationary randomized allocation policy Φ for **P2** that determines feasible allocation strategies $\mathbf{a}_{u,s}^{\Phi}(t), \mathbf{b}_u^{\Phi}(t)$, $\forall u \in \mathcal{A}(t), \forall s \in \mathcal{S}, \forall t$, independent of the current queue backlogs $Q_s(t), \forall s \in \mathcal{S}$ in every time slot t, and yields the following steady state values:

$$\begin{aligned} &\mathbb{E}\{\mathbf{a}_s^{\Phi}(t)\} = \mathbb{E}\left\{ \sum\nolimits_{u \in \mathcal{A}(t)} \mathbf{a}_{u,s}^{\Phi}(t) \right\} = r_s^*, \\ &\mathbb{E}\{\mathbf{a}_s^{\Phi}(t)\big(\mathbf{a}_s^{\Phi}(t) + 2M_s(t) + 1\big)\} \leq \overline{c_d^*}, \\ &\mathbb{E}\left\{ \sum_{u \in \mathcal{A}(t)} \big((1 - \sum_{s \in \mathcal{S}} \mathbf{a}_{u,s}^{\Phi}(t)) h_u\big) \right\} = \overline{c_h^*} \end{aligned} \quad (20)$$

Let $\beta = \omega_r \overline{b_r} - \omega_d \overline{c_d} - \omega_h \overline{c_h}$ denote the system benefit (Section 3.3) in time slot t. For all feasible allocation solutions, which includes those produced by Φ, the proposed

1. www.ibm.com/analytics/cplex-optimizer
2. www.gurobi.com

SUAC algorithm minimizes the drift-minus-benefit bound proven in Lemma 1, thus the following inequality holds:

$$\begin{aligned}&\Delta(Q(t)) - V\,\mathbb{E}\{\beta(t)|Q(t)\}\\&\leq B + \sum_{s\in\mathcal{S}}\mathbb{E}\left\{\mathbf{a}_s^{\Phi}(t)^2 VK + \mathbf{a}_s^{\Phi}(t)P_s(t)|Q(t)\right\}\\&\quad+\sum_{u\in\mathcal{A}(t)}\mathbb{E}\left\{V\omega_h h_u\big(1-\sum_{s\in\mathcal{S}}\mathbf{a}_{u,s}^{\Phi}(t)\big)|Q(t)\right\}\end{aligned}\tag{21}$$

Taking expectations of the above inequality and using the law of iterated expectations give us:

$$\begin{aligned}&\mathbb{E}\{L(Q(t+1))\} - \mathbb{E}\{L(Q(t))\} - V\,\mathbb{E}\{\beta(t)\}\\&\leq B + \sum_{s\in\mathcal{S}}\mathbb{E}\left\{\mathbf{a}_s^{*}(t)^2 VK + \mathbf{a}_s^{*}(t)P_s(t)|Q(t)\right\}\\&\quad+\sum_{u\in\mathcal{A}(t)}\mathbb{E}\left\{V\omega_h h_u\big(1-\sum_{s\in\mathcal{S}}\mathbf{a}_{u,s}^{*}(t)\big)|Q(t)\right\}\end{aligned}\tag{22}$$

By plugging (20) into the right-hand side of (22) and rearranging the terms, we have:

$$\begin{aligned}\mathbb{E}\{L(Q(t+1))\} - \mathbb{E}\{L(Q(t))\} - V\,\mathbb{E}\{\beta(t)\} \leq B - V\beta^*\\+\sum_{s\in\mathcal{S}}\mathbb{E}\{\mathbf{a}_s^*(t)Q_s(t)|Q(t)\}\end{aligned}\tag{23}$$

The above holds for all $t \in \{0, 1, 2, ...\}$. By summing the above inequality over $t \in \{0, 1, ..., T-1\}$ for some integer $T > 0$ and applying the law of telescoping sums, we have:

$$\begin{aligned}\mathbb{E}\{L(Q(T))\} - \mathbb{E}\{L(Q(0))\} - V\sum_{t=0}^{T-1}\mathbb{E}\{\beta(t)\} \leq TB - VT\beta^*\\+\sum_{t=0}^{T-1}\sum_{s\in\mathcal{S}}\mathbb{E}\{\mathbf{a}_s^*(t)Q_s(t)|Q(t)\}\end{aligned}\tag{24}$$

Since $L(Q(0)) = 0$, $L(Q(T)) \geq 0$, and $\mathbf{a}_s^*(t)Q_s(t) \geq 0$, $\forall s \in \mathcal{S}, \forall t$, dividing both side of the above inequality by T yields:

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\{\beta(t)\} \geq \beta^* - \frac{B}{V}\tag{25}$$

The proof of the first part of Theorem 1 completes by letting $T \to \infty$.

Next, we prove the second part of of Theorem 1. Suppose there are constants $B \geq 0$, $V \geq 0$, $\epsilon \geq 0$, and β^* such that for all time slot t and all possible values of $Q(t)$, we have:

$$\Delta(Q(t)) + V\,\mathbb{E}\{\beta(t)|Q(t)\} \leq B + V\beta^* - \epsilon\sum_{s\in\mathcal{S}}Q_s(t)\tag{26}$$

Taking expectations of both sides, summing over $t \in \{0, 1, ..., T-1\}$, and applying the law of iterated expectations yield:

$$\begin{aligned}&\mathbb{E}\{L(Q(T))\} - \mathbb{E}\{L(Q(0))\} + V\sum_{t=0}^{T-1}\mathbb{E}\{\beta(t)|Q(t)\}\\&\leq T(B + V\beta^*) - \epsilon\sum_{t=0}^{T-1}\sum_{s\in\mathcal{S}}\mathbb{E}\{Q_s(t)\}\end{aligned}\tag{27}$$

Assume the expected system benefit $\beta(t)$ is lower bounded by a finite value β^{min} so that we have $\mathbb{E}\{\beta(t)\} \geq \beta^{min}$ for all possible allocation decisions. As $L(Q(0)) = 0$ and $L(Q(T)) > 0$, plugging this into the above inequality and rearranging terms yield:

$$\frac{1}{T}\sum_{t=0}^{T-1}\sum_{s\in\mathcal{S}}\mathbb{E}\{Q_s(t)\} \leq \frac{B + V(\beta^* - \beta^{min})}{T}\tag{28}$$

Letting $T \to \infty$ completes the proof of the second part of Theorem 1. □

5 Evaluation

We have performed a series of experiments to verify and evaluate the performance of our proposed SUAC algorithm.

5.1 Experiment Setup

Edge servers: We use the EUA dataset[3] [6], which contains the geographic locations of end-users and all cellular base stations in Australia. Then, we simulate a highly dense 500m×500m area covered by 26 base stations, assuming each base station is equipped with an edge server. The coverage radius of each edge server is randomly generated within a range of 100-150m. The edge server capacities are randomly generated following a normal distribution $\mathcal{N}(\mu, \sigma^2)$, where μ is the average capacity of each resource type (CPU, RAM, storage, and bandwidth), and the standard deviation $\sigma = 10$ for all experiments conducted in this paper. Since a normal distribution might contain negative numbers, any negative amount of computing resources generated is rounded up to 1. We set $w_s = 1$ for all edge servers $s \in \mathcal{S}$ so that the benefit gained from the same throughput would be equal among all edge servers.

Edge users: All edge servers are able to serve $N = \sum_{s\in\mathcal{S}} N_s$ users simultaneously. The number of newly-arrived users in each time slot $|\mathcal{A}(t)|$ is drawn from a Poisson distribution with rate $[0, \zeta N]$, where $\zeta \in (0, 0.1]$ controls the traffic intensity. We assume $\mathcal{R} = \{CPU, RAM, storage, bandwidth\}$. Each resource requirement is a $|\mathcal{R}|-$dimensional vector, where each vector component is the normalized amount of a resource type in $\mathcal{R}$. Each user's resource requirement is randomly generated using a uniform distribution within $< 1, 1, 1, 1 >$ and $< 4, 4, 4, 4 >$. Each user's session duration is uniformly distributed in [10, 20] time slots. Each time slot is set at 30-second length. The latency experienced by a user, if allocated to the remote cloud, is randomly set in [50,250] ms.

All the experiments are conducted on a Windows machine equipped with Intel Core i5-7400T processor (4 CPUs, 2.4GHz) and 8GB RAM. The optimization problem **P2** is solved with IBM ILOG CPLEX Optimizer[4] in line 3 of Algorithm 1.

5.2 Performance Benchmark

We evaluate SUAC against the state of the art and two baseline approaches:

- **Join-the-Shortest-Queue (JSQ):** The authors of [8], [9] propose a class of randomized algorithms for placing VMs in physical servers that can achieve

3. www.github.com/swinedge/eua-dataset
4. www.ibm.com/analytics/cplex-optimizer

TABLE 2: Experiment sets

	μ	ζ	V	ω_r	ω_d	ω_h
Set #1	40	$0.01, 0.02, ..., 0.1$	0.3	300	7	200
Set #2	$10, 20, ..., 90$	0.07	0.3	300	7	200
Set #3	40	0.07	$0.05, 0.1, 0.15, ..., 0.45$	$200, 300, 400, 500$	7	200
Set #4	40	0.07	$0.05, 0.1, 0.15, ..., 0.45$	300	$3, 5, 7, 9$	200
Set #5	40	0.07	$0.05, 0.1, 0.15, ..., 0.45$	300	7	$50, 200, 350, 500$

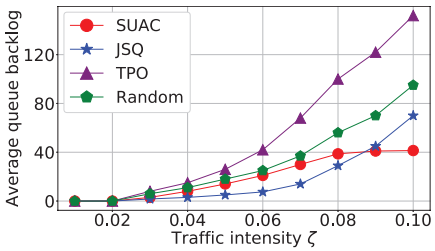


Fig. 2: Average queue backlog vs. varying traffic intensity ζ under four algorithms (Set #1).

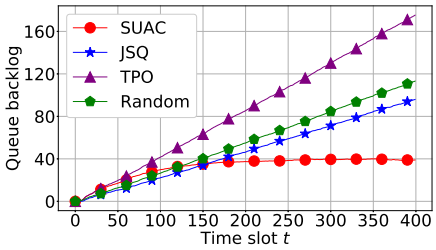


Fig. 3: Queue backlog evolution under four algorithms with $\zeta = 0.3$ (Set #1).

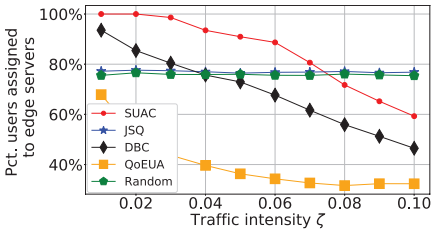


Fig. 4: Percentage of users allocated to edge servers (Set #1).

maximum throughput without preemptions in a cloud computing environment. The users and edge servers in our problem can be seen as VMs and physical servers in their problems. Since our problem has a third tier (the remote cloud server), we modify their approach to incorporate this extension. We consider the remote cloud server as a "virtual" edge server without a queue. Suppose a user u arrives at time slot t, it can be allocated to the cloud with probability $\Pr(\mathbf{b}_u = 1) = \Pr(\mathbf{a}_{u,1} = 1) = ... = \Pr(\mathbf{a}_{u,S} = 1)$. If not allocated to the cloud, it will be allocated to the edge server with the shortest queue. Each queue maintains a Poisson clock to control when to serve the queuing users.

- **Density-based Clustering (DBC):** In [17], users are allocated to cloudlets (equivalent to edge servers in our work) using a density-based clustering algorithm that takes into account the distances between users and their neighbor edge servers. Users are to be allocated to an edge server that has the most candidate users first, i.e., users that are covered by the server. For example, say users u_1 and u_2 can be allocated to either edge servers s_1 or s_2. Edge server s_1 covers more users than edge server s_2. User u_1 is closer to s_1 than it is to s_2, and u_2 is closer to s_2 than it is to s_1. Hence, the users allocation will start from s_1 since it has more candidate users. User u_1 has a higher allocation priority than u_2 when being assigned to s_1. Since this approach does not take queuing system into account, we assume that excessive users, i.e., users that are allocated to edge servers but the edge servers do not have sufficient computing resources to serve the users, will be redirected to the remote cloud.
- **QoE-aware User Allocation (QoEUA):** In [7], the authors map each user resource requirement to a quality-of-experiment (QoE) level. They optimally allocate users to edge servers using an integer programming approach so that the total QoE of all the users is maximized. Similar to DBC, this approach does not take queuing system into account. Thus, we assume that excessive users will be redirected to the remote cloud.
- **Throughput Optimal (TPO):** To achieve the maximum throughput benefit, this approach completely ignores the remote cloud. Every user will be allocated to the edge server with the shortest queue. Predictably, this approach may well result in a high queuing delay cost.
- **Random:** New users are uniformly allocated to either the cloud or edge servers at random.

5.3 Experiment Sets

We conduct a series of experiments with different varying parameters to analyze the performance of SUAC in various EC scenarios. Table 2 summarizes all the experiment sets which will be discussed in the next section. We first experiment with various traffic intensities (ζ) in Set #1, ranging from light traffic to very intense traffic, to simulate different user arrival rates. In Set #2, we vary the computing resource capacity (μ) that each edge server has to serve users, ranging from scarce resources to abundant resources; this impacts the service rate of each edge server. After that, we vary the control parameter V used in Eq. (17), and at the same time, the throughput benefit weight ω_r, queuing delay cost weight ω_d, and latency cost weight ω_h in Sets #3, #4, and #5, respectively. These domain-specific parameters are used in Eq. (17) to indicate the priorities for SUAC's pursuit of system throughput, queuing delay, and latency, respectively, when SUAC allocates users to edge servers over time. In a real-world setting, a service provider can determine those parameters according to their needs. Note that in those three experiment sets, parameters $V, \omega_r, \omega_d, \omega_h$ have no impact on the allocation results produced by the five benchmark approaches since they work independently of those parameters. Each experiment setting is executed for

a duration of 1,000 time slots.

5.4 Experiment Results

5.4.1 Impact of Traffic Intensity (User Arrival Rate)

In Set #1, we vary the traffic intensity ζ from $0.01, 0.02$ to as high as 0.1. For example, say the 26 edge servers can serve 500 users simultaneously, $\zeta = 0.1$ means that there are 50 new users joining the system in every time slot on average. Fig. 2 illustrates the time-average queue backlog under four approaches when the EC system is put under various traffic intensities. The time-average queue backlog is the average number of users queuing per server during the simulated duration. Fig. 3 depicts the evolution of the queue backlog in Set #1 when $\zeta = 0.3$ during the first 400 time slots. Note that DBC and QoEUA are not shown in those two figures because they do not employ a queuing system, or have no queues.

The time-average queue backlogs under JSQ, TPO, and Random (Fig. 2) grow linearly with the traffic intensity. At $\zeta = 0.1$, the average queue backlog of TPO is very long, i.e., roughly 140 users queuing per server, which is unacceptable for any latency-sensitive services. This is unsurprising since TPO only allocates all users to edge servers. On the other hand, under SUAC, the time-average queue backlog slowly increases with the increasing traffic intensity, then converges and remains unchanged at approximately 40 users regardless of the varying traffic intensity ($\zeta = 0.08 - 0.1$). A service provider can easily increase or decrease the queue backlog level by increasing or decreasing the control parameter V. In Fig. 3, the queue backlogs of all four approaches gradually increase during the first few time slots. After that, while the other three approaches keep getting their queues more congested, SUAC stops allocating too many users to edge servers and starts directing them to the remote cloud, stabilizing the EC system instead of overloading it. We noticed the same phenomenon in all other experiments, which are not presented here for brevity. This strongly demonstrates the ability of SUAC to stabilize the EC system under any traffic conditions.

Fig. 4 shows the percentage of users allocated to edge servers (the higher the better). The numbers of users allocated to edge servers by SUAC, DBC, and QoE decrease as we increase the traffic intensity ζ. This is expected because the edge servers can only serve up to a certain number of users and the rest have to be allocated to the remote cloud. Otherwise, the queues would be heavily congested. The numbers of users allocated to edge servers by JSQ and Random remain constant since they decide if a user is allocated to the remote cloud by using the same randomness factor. Under some experiment settings ($\zeta = 0.08 - 0.1$), JSQ and Random manage to allocate more users to edge servers than SUAC. However, SUAC still beats them in terms of system benefit because under JSQ and Random, the queuing delay cost outweighs the throughput benefit. Under all other experiment settings, SUAC significantly outperforms all other approaches. TPO is not presented here because it does not allocate users to the remote cloud.

Fig. 5 visualizes the normalized time-average system benefit gained by the six approaches under different traffic intensities. At the very beginning, all four approaches achieve relatively equal performance since the traffic was very light, thus all the users could be allocated to edge servers without queuing. As more users arrive, SUAC starts to significantly outperform the other three approaches since they suffer from a very high queuing delay cost (JSQ, TPO, and Random) or cloud latency cost (DBC and QoEUA). The throughput benefit and queuing delay cost produced by SUAC remain unchanged when increasing traffic intensity since all the queues are kept stabilized as discussed above. To stabilize the system under intense traffic conditions, SUAC directs new users to the remote cloud, hence the considerable increase in the cloud latency cost, which in turn results in the loss of system benefit. We can infer that in order to deal with an increasing user arrival rate, a service provider needs to hire more edge computing resources to increase the service rate.

5.4.2 Impact of Edge Server Capacity (Service Rate)

In this section, we evaluate the impact of service rate (Set #2), which is determined by the amount of computing resources available on edge servers. An increase in the average edge server capacity eventually leads to an increase in the service rate of the system, meaning an edge server can hold more users in its queue without increasing the queuing delay. This is demonstrated in Fig. 6, as the average server capacity μ increases from 10 to 30, the average queue backlog under SUAC also gradually increases, starting from around 10 users to almost 40 users per edge server's queue. In other words, SUAC can allocate more users to edge servers only when it is safe to do so given the current service rate. From $\mu = 30$ onward, its average queue backlog steadily decreases since the service rate is now relatively high, which allows edge servers to serve more users simultaneously. DBC and QoEUA do not employ a queuing system so they are presented in this figure.

In contrast, the other three approaches (JSQ, TPO, and Random) work independently of the service rate. As a result, increasing the service rate will lessen the stress on the queue backlogs under those approaches. In the figure, we can see that the average queue backlogs of JSQ, TPO, and Random gradually decrease as the average server capacity increases. We can predict that when all the edge servers have an abundant amount of computing resources, the queue backlogs under all four approaches will eventually converge to close to zero. However, as aforementioned, such resource-abundant situations are extremely unlikely to happen in the EC environment. In such cases, joint load balancing (i.e., SUAC) and dynamic management of EC resources, or any other simple approaches, can actually control queue backlog effectively.

Fig. 7 visualizes the corresponding normalized time-average system benefit in the experiment analyzed above. Again, SUAC clearly outperformed all other approaches under any experiment setting. For the same reason discussed above, the time-average system benefit of all approaches will eventually converge once the service provider is able to hire an excessive amount of edge computing resources, which would be highly expensive and unlikely to happen in any real-world scenarios. JSQ and Random have a randomness factor so there will always be some users allocated to the cloud despite a large amount of computing resources. DBC does not employ a queuing system so some users will

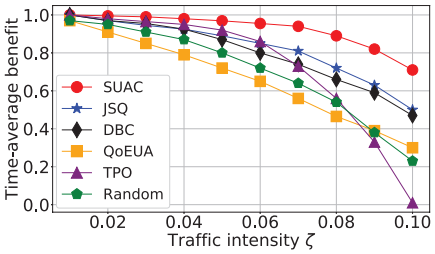


Fig. 5: Time-average system benefit vs. varying traffic intensity ζ under four algorithms (Set #1).

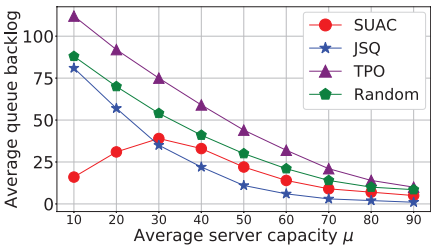


Fig. 6: Average queue backlog vs. varying server capacity μ under four algorithms (Set #2).

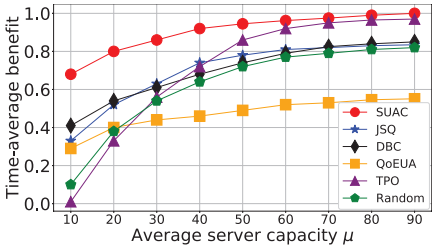


Fig. 7: Time-average system benefit vs. varying server capacity μ under four algorithms (Set #2).

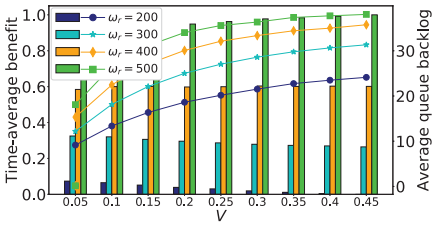


Fig. 8: Time-average system benefit vs. varying values of V under SUAC with different values of ω_r (Set #3).

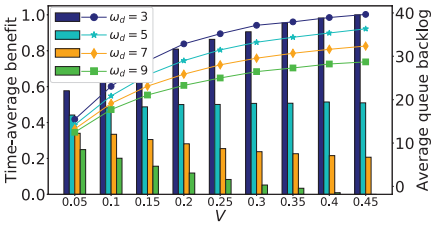


Fig. 9: Time-average system benefit vs. varying values of V under SUAC with different values of ω_d (Set #4).

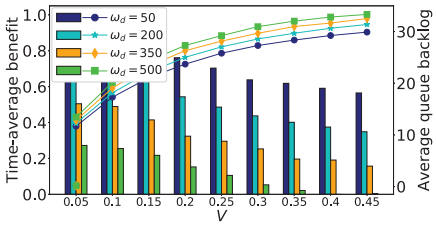


Fig. 10: Time-average system benefit vs. varying values of V under SUAC with different values of ω_h (Set #5).

be allocated to the cloud straightaway, even though they could have just waited in a queue for a short period of time. Similar to DBC, QoEUA does not employ a queuing system. Given those rationales, JSQ, Random, DBC, and QoE do not perform as good as SUAC or TPO even when the average service capacity is large.

5.4.3 *Impact of Control Parameter V and Associated Weights ω_r, ω_d, and ω_h*

In this section, we investigate the impact of the trade-off control parameter V as well as other associated weights, namely ω_r, ω_d, and ω_h. Since V, ω_r, ω_d, and ω_h do not influence the user allocation decisions made by JSQ, TPO, and Random, we do not include those approaches in this section. The effectiveness of SUAC against them has already been experimentally analyzed in Sections 5.4.1 and 5.4.2.

In Sets #3, #4, and #5, we simulate varying values of V under SUAC with different values of throughput benefit weight ω_r (Set #3), queuing delay cost weight ω_d (Set #4), and cloud latency cost weight ω_h (Set #5). In Figs. 8, 9, and 10, the y-axis on the left-hand side corresponds to the value of the bar graph, and the y-axis on the right-hand side corresponds to the value of the line graph. As expected, a higher value of V results in a longer average queue backlog as can be seen in all experiment sets. This demonstrates the flexibility of SUAC that enables service providers to control the congestion state of their EC systems.

The sensitivity of throughput benefit weight ω_r (Set #3). Fig. 8 plots the average queue backlog and time-average system benefit gained by SUAC with different values of ω_r. A higher ω_r means that the service provider places a higher priority on the benefit gained from system throughput. As a result, SUAC attempts to allocate more users to edge servers, leading to a higher queue backlog (more expensive queuing delay cost). Since more users are being allocated to edge servers, fewer users will be allocated to the remote cloud, lowering the cloud latency cost and balancing out the queuing delay cost. As the cloud latency cost and queuing delay cost are balancing each other, a greater throughput benefit weight ω_r will eventually gain a higher time-average system benefit under the same value of V.

In this experiment set, the time-average system benefit under the same ω_r remains virtually unchanged across varying values of V even when the queue backlog changes. The reason is that the increasing throughput benefit, increasing queuing delay cost and decreasing cloud latency cost are balancing out each other. This will not be the case if the service provider adjusts ω_d or ω_h. Sets #4 and #5 will demonstrate how the changing patterns of time-average system benefit with changing V can be influenced differently compared to the pattern observed in this experiment set.

The sensitivity of queuing delay cost weight ω_d (Set #4). Fig. 9 plots the average queue backlog and time-average system benefit gained by SUAC with different values of ω_d. A higher value of ω_d means that a more congested queue is penalized harder than a less congested queue. This is why under the same V, a lower ω_d results in a higher average queue backlog, hence a higher throughput benefit, a lower latency cost, and a higher queuing delay cost. Collectively, a lower ω_d gains a higher system benefit.

In terms of the time-average system benefit with V changing, there are several patterns here. With $\omega_d = 3$, the time-average system benefit tends to increase with the increasing V because the increasing throughput benefit far outweighs the decreasing queuing delay and latency costs. With $\omega_d = 5$, they balance out each other so there is not much difference across different values of V. With $\omega_d = 7$ and 9, the increasing throughput benefit starts being outweighed by the decreasing queuing delay and latency costs, hence the decrease in time-average system benefit. Again,

those patterns can be changed if the service provider adjusts the values of ω_h or ω_r.

The sensitivity of latency cost weight ω_h (Set #5). Fig. 10 plots the average queue backlog and time-average system benefit gained by SUAC with different values of ω_h. A higher value of ω_h means that the more users allocated to the remote cloud, the harder the penalty is. Therefore, with the same V, SUAC will lean towards allocating more users to edge servers under a higher value of ω_h, leading to a longer average queue backlog, which also incurs a higher queuing delay cost. Since the latency cost of a higher ω_h is much more expensive than that of a lower ω_h, the system benefit gained by a higher ω_h is lower.

As V increases, the time-average system benefit gained by SUAC decreases for all experimental values of ω_h. This occurs because the increasing queuing delay cost outweighs the increasing throughput benefit and decreasing latency cost. Similar to Sets #3 and #4, the pattern can be adjusted with a different value of ω_r or ω_d.

As demonstrated, the control parameter V, and weight parameters ω_r, ω_d, and ω_h control the congestion state of an EC system, which in turn influence the system benefit. Those parameters are selected by the service provider depending on their needs, or the significance of the throughput benefit, queuing delay cost, and latency cost. Given those pre-selected parameters, SUAC guides the user allocation process over time so that the system benefit is maximized.

6 Related work

In the EC environment, applications are deployed on edge servers so that users can access with low latency. From a service provider's perspective, the user allocation problem is the problem of how to allocate its users to proper edge servers so that some optimization objectives are achieved.

Jia et al. [17] study the users-to-cloudlets allocation and cloudlet placement problems in wireless metropolitan networks. In their scenario, users are connected to access points, which might or might not have cloudlets, using a density-based clustering algorithm. They do not consider user arrivals and departures. Instead, they assume a fixed number of users and vary the task offloading rate of each user. The user allocation problem in [18] is more of a base-stations-to-edge-clouds allocation problem as opposed to our users-to-base-stations (edge servers) allocation problem. Ouyang et al. [19] study the service placement problem that takes into account user mobility. The user-to-edge server allocation is assumed to be automatically handled by the edge infrastructure provider. The authors of [5] also consider user mobility and attempt to minimize the number of user reallocations when a user moves across the coverage of different edge servers. However, their approach allocates users to only edge servers without taking remote cloud servers or queuing system into consideration. In [14], the authors assume that an edge server can serve a user outside its coverage via an intermediary server when that intermediary server is being overloaded in terms of computation capability (but still has sufficient communication capability to handle the user). In real-world scenarios, this assumption is not necessarily realistic. Even if it is, it is only applicable to scenarios without many users. In most real-world scenarios, due to edge servers' constrained computing resources, it is unlikely for them to serve their neighbor servers' users. Most of the time, they will be busy serving their own users (those that are covered by the edge server itself). Thus, SUAC chooses to allocate users that cannot be served on time by edge servers to the remote cloud.

In [20], a one-user-one-VM (virtual machine) scenario is addressed. This might be impractical in an EC system since launching a VM is a time-consuming process, which defeats the edge computing's purpose of ensuring a low-latency connection. There is a line of work on throughput-optimal VM scheduling in cloud computing [8], [9], [21], [22]. These works, however, consider a two-tier scenario (VM - physical server) and thus are not directly applicable in our three-tier EC scenario (user - edge server - cloud).

Lai et al. [6] tackle the user allocation problem in a static EC system with the goal of minimizing the number of edge servers that a service provider needs to hire to serve its users. Avgeris et al. [23] also aim to minimize the number of active servers in a computation offloading problem. The aforementioned works are impractical in a dynamical scenario, where numerous users come and go in every time slot, and a lot of those users are covered by only one edge server. This, plus the resource scarcity of edge servers, makes it nearly impossible to not use all the given edge servers in any time slot. In [7], the authors attempt to allocate users to edge servers in a scenario where a user's quality of service (which corresponds to the user's computing resource requirement) can be dynamically adjusted, which is not supported in our scenario. Their objective is to maximize user satisfaction. Qiang et al. [4] propose a game-theoretic user allocation approach that minimizes system costs, measured by the amount of computing resources needed to serve users and the penalty of having unallocated users. They calculate the computing resource consumption using a model that is different from ours, hence being inapplicable to our scenario. In all those works, the EC system is assumed to be static and they do not consider user arrivals and departures over time, which is common and critical in edge computing.

Computation offloading is an important research track in edge computing that shares some similarities with the user allocation problem. Nevertheless, those two problems are differentiated by several essential characteristics. In the computation offloading problem, a user generates a series of computation tasks, which can be partly executed on its local device and edge servers (partial offloading), or completely on edge servers or remote clouds (full offloading) [24]. A computation task usually has a single-dimensional resource requirement (CPU cycles) [3], [10], [16], [25]. Nevertheless, in real-world scenarios, a service provider needs to dedicate multiple types of resources to serve a user on an edge server [2], [4], [26]. In addition, each computation task is usually assumed to be small and thus can be completed in a known duration less than a single time slot [3], [16], [27], or within a short time frame [26]. In [13], the duration of a job spans across multiple time slots but it is known in advance. In a real EC system, the duration of a user session is unknown and likely to be longer than one time slot. Chen et al. [28] only consider a single-server scenario while a real-world EC scenario usually involves multiple edge servers. In some

works [3], [16], users are assumed to be pre-allocated to edge servers before proceeding to the task offloading phase. Moreover, a user in the user allocation problem must be allocated to a server, either an edge server or a cloud server, instead of being able to partially offload its computation tasks, or share computation tasks among edge servers.

In terms of the employed approach, Lyapunov optimization has been proven to be very effective and widely applied when dealing with a time-slotted scenario [13], [16], [19], [27], [29]. Previous works that employ the standard Lyapunov optimization framework usually assume that each job or task can be completely executed within the duration of a time slot [3], [16], [27]. We apply the Lyapunov optimization in a more general situation where a user session can last longer than one time slot, which could be a few seconds or minutes in a highly stochastic EC environment.

7 Conclusion and Future Work

In this paper, we investigate the stochastic edge user allocation problem in a time-slotted EC system. A service provider needs to take into account several factors such as queuing delay and cloud latency costs while maximizing the system throughput. We address a realistic EC environment where users come and go dynamically and randomly over time, making it hard to find an optimal allocation due to the lack of future information. We propose SUAC – a Lyapunov optimization-based online algorithm that allocates users without requiring any information about user arrivals and departures. As theoretically proven, SUAC achieves a bounded performance guarantee. We conduct a series of experiments based on a real-world dataset, which clearly demonstrates the superiority of SUAC over the existing approaches and its ability to achieve a desired tradeoff as well as strongly stabilize the system.

There are several research directions that can be addressed in the future. In this paper, the optimization problem in every time slot is solved optimally. If the size of the problem massively scales up, a more efficient approach for allocating users in individual time slots is needed. In addition, one could consider the channel interference when allocating users in a high density EC system to improve user experience further.

Acknowledgments

This research is partly funded by Australian Research Council Discovery Project grants DP170101932 and DP180100212. Grundy is supported by Laureate Fellowship FL190100035.

References

[1] D. Wang, Y. Peng, X. Ma, W. Ding, H. Jiang, F. Chen, and J. Liu, "Adaptive wireless video streaming based on edge computing: Opportunities and approaches," *IEEE Transactions on Services Computing*, vol. 12, no. 5, pp. 685–697, 2018.

[2] Y. Liang, G. Jidong, S. Zhang, J. Wu, Z. Tang, and B. Luo, "A utility-based optimization framework for edge service entity caching," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 11, pp. 2384–2395, 2019.

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

[4] Q. He, G. Cui, X. Zhang, F. Chen, S. Deng, H. Jin, Y. Li, and Y. Yang, "A game-theoretical approach for user allocation in edge computing environment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 3, pp. 515–529, 2020.

[5] Q. Peng, Y. Xia, Z. Feng, J. Lee, C. Wu, X. Luo, W. Zheng, H. Liu, Y. Qin, and P. Chen, "Mobility-aware and migration-enabled online edge user allocation in mobile edge computing," in *Proceedings of IEEE International Conference on Web Services*. IEEE, 2019, pp. 91–98.

[6] P. Lai, Q. He, M. Abdelrazek, F. Chen, J. Hosking, J. Grundy, and Y. Yang, "Optimal edge user allocation in edge computing with variable sized vector bin packing," in *Proceedings of International Conference on Service-Oriented Computing*. Springer, 2018, pp. 230–245.

[7] P. Lai, Q. He, G. Cui, X. Xia, M. Abdelrazek, F. Chen, J. Hosking, J. Grundy, and Y. Yang, "Edge user allocation with dynamic quality of service," in *Proceedings of International Conference on Service-Oriented Computing*. Springer, 2019, pp. 86–101.

[8] J. Ghaderi, "Randomized algorithms for scheduling vms in the cloud," in *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2016, pp. 1–9.

[9] K. Psychas and J. Ghaderi, "Randomized algorithms for scheduling multi-resource jobs in the cloud," *IEEE/ACM Transactions on Networking*, no. 99, pp. 1–14, 2018.

[10] Z. Hong, W. Chen, H. Huang, S. Guo, and Z. Zheng, "Multi-hop cooperative computation offloading for industrial IoT–edge–cloud computing environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 12, pp. 2759–2774, 2019.

[11] F. Guo, H. Zhang, H. Ji, X. Li, and V. C. Leung, "An efficient computation offloading management scheme in the densely deployed small cell networks with mobile edge computing," *IEEE/ACM Transactions on Networking*, vol. 26, no. 6, pp. 2651–2664, 2018.

[12] A. Morgado, K. M. S. Huq, S. Mumtaz, and J. Rodriguez, "A survey of 5G technologies: Regulatory, standardization and industrial perspectives," *Digital Communications and Networks*, vol. 4, no. 2, pp. 87–97, 2018.

[13] X. Wang, K. Wang, S. Wu, S. Di, H. Jin, K. Yang, and S. Ou, "Dynamic resource scheduling in mobile edge cloud with cloud radio access network," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 11, pp. 2429–2445, 2018.

[14] B. Gao, Z. Zhou, F. Liu, and F. Xu, "Winning at the starting line: Joint network selection and service placement for mobile edge computing," in *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2019, pp. 1459–1467.

[15] 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, 2010.

[16] 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.

[17] M. Jia, J. Cao, and W. Liang, "Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks," *IEEE Transactions on Cloud Computing*, vol. 5, no. 4, pp. 725–737, 2015.

[18] Y. Guo, S. Wang, A. Zhou, J. Xu, J. Yuan, and C.-H. Hsu, "User allocation-aware edge cloud placement in mobile edge computing," *Software: Practice and Experience*, vol. 50, no. 5, pp. 489–502, 2019.

[19] T. Ouyang, 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, no. 10, pp. 2333–2345, 2018.

[20] L. Li, Q. Guan, L. Jin, and M. Guo, "Resource allocation and task offloading for heterogeneous real-time tasks with uncertain duration time in a fog queueing system," *IEEE Access*, vol. 7, pp. 9912–9925, 2019.

[21] S. T. Maguluri and R. Srikant, "Scheduling jobs with unknown duration in clouds," in *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2013, pp. 1887–1895.

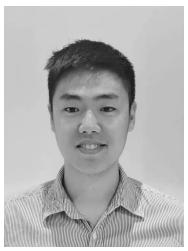
[22] ——, "Scheduling jobs with unknown duration in clouds," *IEEE/ACM Transactions On Networking*, vol. 22, no. 6, pp. 1938–1951, 2013.

[23] M. Avgeris, D. Dechouniotis, N. Athanasopoulos, and S. Papavassiliou, "Adaptive resource allocation for computation offloading: A control-theoretic approach," *ACM Transactions on Internet Technology*, vol. 19, no. 2, pp. 1–20, 2019.

[24] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.
[25] Y. Mao, J. Zhang, S. Song, and K. B. Letaief, "Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems," *IEEE Transactions on Wireless Communications*, vol. 16, no. 9, pp. 5994–6009, 2017.
[26] M. Hu, L. Zhuang, D. Wu, Y. Zhou, X. Chen, and L. Xiao, "Learning driven computation offloading for asymmetrically informed edge computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 8, pp. 1802–1815, 2019.
[27] F. Liu, Z. Zhou, H. Jin, B. Li, B. Li, and H. Jiang, "On arbitrating the power-performance tradeoff in saas clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 10, pp. 2648–2658, 2013.
[28] 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, 2015.
[29] W. Chen, D. Wang, and K. Li, "Multi-user multi-task computation offloading in green mobile edge cloud computing," *IEEE Transactions on Services Computing*, vol. 12, no. 5, pp. 726–738, 2018.

Phu Lai received his MSc degree in Information Technology in 2017 and is currently working toward a PhD degree at Swinburne University of Technology, Australia. His research interests include software engineering, cloud computing and edge computing.

Qiang He received the first PhD degree from Swinburne University of Technology (SUT), Australia, in 2009 and the second PhD degree in computer science and engineering from Huazhong University of Science and Technology (HUST), China, in 2010. He is a lecturer at Swinburne University of Technology. His research interests include software engineering, cloud computing, and services computing. More details about his research can be found at www.sites.google.com/site/heqiang/.



Xiaoyu Xia received his Master degree from The University of Melbourne, Australia in 2015. He is a PhD candidate at Deakin University, Australia. His research interests include edge computing, service computing and software engineering.

Feifei Chen received her PhD degree from Swinburne University of Technology, Australia, in 2015. She is a lecturer at Deakin University. Her research interests include software engineering, cloud computing and green computing.

Mohamed Abdelrazek is an Associate Professor of Software Engineering and IoT at Deakin University. Mohamed has more than 15 years of the software industry, research, and teaching experience. Before joining Deakin University in 2015, he worked as a senior research fellow at Swinburne University of Technology and Swinburne-NICTA software innovation lab (SSIL). More details about his research can be found at www.sites.google.com/site/mohamedalmorsy/.

John Grundy is the Senior Deputy Dean for the Faculty of Information Technology and a Professor of Software Engineering at Monash University. He is a Fellow of Automated Software Engineering, Fellow of Engineers Australia, Certified Professional Engineer, Engineering Executive, Member of the ACM and Senior Member of the IEEE. His current interests include large-scale systems engineering, software engineering education, etc. More details about his research can be found at www.sites.google.com/site/johncgrundy/.

John Hosking is Dean of Science at the University of Auckland and Adjunct Professor of Computer Science at the ANU. His research interests are primarily in the Software Engineering/Software Tools area and he is an active member of the Automated Software Engineering and Visual Languages research communities. John is a Fellow of the Royal Society of New Zealand and a Member of the Ako Aotearoa Academy of Tertiary Teaching Excellence.

Yun Yang received his PhD degree from the University of Queensland, Australia, in 1992. He is a full professor at Swinburne University of Technology. His research interests include software engineering, cloud and edge computing, workflow systems, and service-oriented computing.