

Northumbria Research Link

Citation: Wang, Xinhou, Wang, Kezhi, Wu, Song, Di, Sheng, Jin, Hai, Yang, Kun and Ou, Shumao (2018) Dynamic Resource Scheduling in Mobile Edge Cloud with Cloud Radio Access Network. IEEE Transactions on Parallel and Distributed Systems, 29 (11). pp. 2429-2445. ISSN 1045-9219

Published by: IEEE

URL: <https://doi.org/10.1109/TPDS.2018.2832124>
<<https://doi.org/10.1109/TPDS.2018.2832124>>

This version was downloaded from Northumbria Research Link:
<http://nrl.northumbria.ac.uk/id/eprint/34137/>

Northumbria University has developed Northumbria Research Link (NRL) to enable users to access the University's research output. Copyright © and moral rights for items on NRL are retained by the individual author(s) and/or other copyright owners. Single copies of full items can be reproduced, displayed or performed, and given to third parties in any format or medium for personal research or study, educational, or not-for-profit purposes without prior permission or charge, provided the authors, title and full bibliographic details are given, as well as a hyperlink and/or URL to the original metadata page. The content must not be changed in any way. Full items must not be sold commercially in any format or medium without formal permission of the copyright holder. The full policy is available online: <http://nrl.northumbria.ac.uk/policies.html>

This document may differ from the final, published version of the research and has been made available online in accordance with publisher policies. To read and/or cite from the published version of the research, please visit the publisher's website (a subscription may be required.)

Dynamic Resource Scheduling in Mobile Edge Cloud with Cloud Radio Access Network

Xinhou Wang, Kezhi Wang, Song Wu, *Member, IEEE*, Sheng Di, *Senior Member, IEEE*, Hai Jin, *Senior Member, IEEE*, Kun Yang, *Member, IEEE*, Shumao Ou, *Member, IEEE*

Abstract—Nowadays, by integrating the *cloud radio access network* (C-RAN) with the *mobile edge cloud computing* (MEC) technology, mobile service provider (MSP) can efficiently handle the increasing mobile traffic and enhance the capabilities of mobile devices. But the power consumption has become skyrocketing for MSP and it gravely affects the profit of MSP. Previous work often studied the power consumption in C-RAN and MEC separately while less work had considered the integration of C-RAN with MEC. In this paper, we present a unifying framework for the power-performance tradeoff of MSP by jointly scheduling network resources in C-RAN and computation resources in MEC to maximize the profit of MSP. To achieve this objective, we formulate the resource scheduling issue as a stochastic problem and design a new optimization framework by using an extended Lyapunov technique. Specially, because the standard Lyapunov technique critically assumes that job requests have fixed lengths and can be finished within each decision making interval, it is not suitable for the dynamic situation where the mobile job requests have variable lengths. To solve this problem, we extend the standard Lyapunov technique and design the *VariedLen* algorithm to make online decisions in consecutive time for job requests with variable lengths. Our proposed algorithm can reach time average profit that is close to the optimum with a diminishing gap ($1/V$) for the MSP while still maintaining strong system stability and low congestion. With extensive simulations based on a real world trace, we demonstrate the efficacy and optimality of our proposed algorithm.

Index Terms—Cloud radio access network; Mobile edge computing; Power-performance tradeoff; Lyapunov optimization; Scheduling.

1 INTRODUCTION

NOWADAYS, in order to meet the mobile traffic demand generated by increasing mobile devices, the existing cellular network is facing high pressure to improve the capacity by building more *base stations* (BSes) [1]. However, due to rapid technological changes in competitive marketplace, *mobile service providers* (MSPs) are challenged with deployment of traditional BS [2]. For example, the MSP needs to spend very high cost to deploy a new BS even though the revenues gained from the increasing requests are very low.

Cloud radio access network (C-RAN) has been proposed to address this challenge and received significant attention in both academia and industry [3], [4]. C-RAN divides the traditional BS into three parts, i.e., *remote radio heads* (RRHs), *baseband unit* (BBU) pool, and the fronthaul link [2]. In C-RAN, RRHs only need to compress and forward the received signals from mobile devices and transmit them to the BBU pool while most of the intensive network computational tasks, such as baseband signal processing, precoding matrix calculation, channel state information estimation are moved to the BBU pool.

However, resource-hungry applications such as face recognition and gaming appeared in our daily life, give resource-constrained and battery-limited mobile devices much pressure [5].

Mobile cloud computing (MCC) has been proved as a promising approach to address such a challenge [6], [7]. MCC augments the capabilities of mobile devices by offloading tasks to the powerful platforms in the cloud. Normally, public clouds (e.g., Google Compute Engine [8] and Amazon EC2 [9]) are used to form the mobile cloud platform. However, such kind of remote public clouds may suffer from long latency due to data transmission through *wide area network* (WAN) [10].

By pushing the cloud into the edge of the network, *mobile edge cloud computing* (MEC) [11] has been proposed to tackle the limitations of MCC. As shown in Fig. 1, MEC can provide cloud resources at the edge of the network that is close to mobile users. The edge cloud provides resource-rich cloud computing infrastructures deployed by MSPs (e.g., AT&T and China Mobile) [10]. In this way, not only can MSP handle the increasing mobile traffic by using C-RAN technology, but it can also enhance the capabilities of mobile devices with the powerful edge cloud. Although cloud computing for both access network (i.e., C-RAN) [3], [4], [12] and end devices (i.e., MEC) [10], [13], [14] has been largely studied, these two important areas have traditionally been addressed separately in the literatures. The research of integration of C-RAN with MEC is still a gap.

However, the latency caused by computation in MEC and network in C-RAN both affect customers' experience [15]. For example in Fig. 1, when a mobile user offloads a job to the edge cloud, the system needs to allocate both network and computation resources. If the system allocates high wireless bandwidth and few computation resources, the job would be transferred into the edge cloud very fast but takes long time to execute thus incurring high latency, vice versa. If the system allocates both high wireless bandwidth and too many computation resources, the job request would obtain its result very fast, but the system can

- X. Wang, S. Wu and H. Jin are with the Services Computing Technology and System Lab, Cluster and Grid Computing Lab in the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China. E-mails: {xwang, wusong, hjin}@hust.edu.cn. The Corresponding Author is Song Wu.
- K. Wang is with Department of Computer and Information Sciences, Northumbria University, UK. E-mail: kezhi.wang@northumbria.ac.uk.
- S. Di is with Argonne National Laboratory, USA. E-mail: sdi1@anl.gov.
- K. Yang is with University of Essex, UK. E-mails: kunyang@essex.ac.uk.
- S. Ou is with Oxford Brookes University, UK. E-mail: sou@brookes.ac.uk.

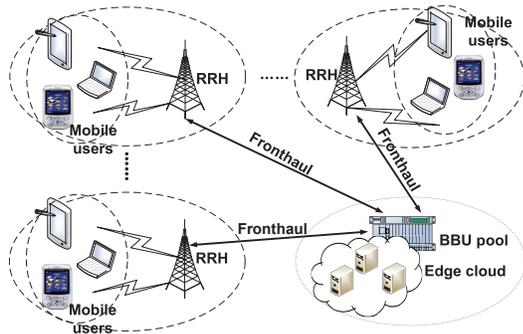


Fig. 1: The overview of mobile system with C-RAN and MEC.

only accommodate few jobs and gain little profit. Therefore, it is necessary to jointly consider these technologies for MSP.

For MSP, on the one hand, the electricity cost of power consumption has become skyrocketing [2]. For example, China Mobile has to spend more than one billion dollars for the electricity every year [2]. Hence, a facing problem of MSP is to minimize the power consumption of the whole system. On the other hand, as analyzed with a real world mobile usage trace in Sec. 2.1, the arrival of job requests from mobile devices are always dynamic and unpredictable. In addition, the jobs from different users usually have variable lengths. Such kind of mobility feature introduces huge challenges for the scheduling of both computation and network resources, leading to fluctuating revenues for MSP over time [16].

Under the unpredictable job requests from mobile users and the skyrocketing electricity cost of power consumption, the objective of the mobile system is to maximize the profit of MSP by scheduling the fronthaul links to accept as many requests as possible (i.e., increasing throughput) while minimizing the power consumption of fronthaul links in C-RAN and servers in edge cloud. In order to optimize such a tradeoff between performance and power, the mobile system needs to tackle the following scheduling challenges: (1) how to schedule each fronthaul link by turning to *active* state for transmitting requests into the BBU pool and *sleep* state to decline users' requests for fronthaul power conservation; (2) how to dispatch the received requests from different users to its corresponding containers in different servers in the edge cloud; (3) how to schedule each container to *running* state¹ for requests processing or *shutdown* state for power conservation.

To tackle the above-mentioned challenges, we apply the Lyapunov technique [18] to design an unifying optimization framework which makes decisions for the fronthaul links, BBU Dispatcher and servers in edge cloud independently and concurrently, solely based on the current system state. Specifically, we have designed (1) a threshold-based scheduling policy for the fronthaul links to improve the throughput as much as possible while guaranteeing system stable; (2) a load balancing policy for request dispatching in the BBU Dispatcher to reduce the delays of the admitted job requests; and (3) an optimal scheduling policy to guide the containers when to keep *shutdown* for power conservation and how to process job requests more efficiently.

¹In docker [17], one can use the command *docker create/run* to create a container. After that, one can use the command *docker start* to start a container to *up* state (i.e., *running* state) for processing requests, or turn to *down* state (i.e., *shutdown* state) by using *docker stop*. Those *down* state containers will not consume resources and power. For convenience, we use *running (shutdown)* state to replace the *up (down)* state for the whole paper.

Note that the standard Lyapunov optimization framework [18] critically assumes that job requests have fixed lengths and can be finished within each decision making interval. However, as analyzed with the real world mobile usage trace in Sec. 2.1, mobile jobs from different users always have variable lengths which may even exceed a time slot. A highlight of this paper is that we can allow a job request with length longer than the time required for an online decision making. In this way, the decisions in consecutive time intervals are strongly correlated while the standard Lyapunov technique cannot handle [18], [19]. By extending the standard Lyapunov technique, we design an algorithm, *VariedLen*, to make online decisions in consecutive time for job requests with variable lengths.

Our main contributions can be summarized as follows:

- We present an unifying optimization framework for maximizing the profit of MSP which manages both network system (i.e., C-RAN) and computing system (i.e., edge cloud).
- By using Lyapunov technology, we design efficient policies for joint optimization of fronthaul link scheduling, requests dispatching and cloud servers scheduling, which can efficiently handle unpredictable mobile job requests. In particular, unlike the standard Lyapunov technology, we allow job requests' lengths to be longer than the length of online decision making, such that the decisions in consecutive time slots are strongly correlated. By extending the standard Lyapunov technique, we design the *VariedLen* algorithm to make online decisions in consecutive time slots for job requests with variable lengths.
- With extensive evaluations based on a real world mobile app usage trace, we demonstrate that the time average profit gained by the *VariedLen* is close to the optimum with a diminishing gap ($1/V$) for MSP while the system stability is still strong and the congestion is low for mobile users.

The organization of this paper is as follows. We propose the power-performance tradeoff model in Sec. 2 and design the *VariedLen* algorithm to dynamically schedule all resources in the mobile system for profit maximization in Sec. 3. We evaluate the performance of our proposed algorithms in Sec. 4 and discuss the related work in Sec. 5. Finally, we conclude our work and discuss the future work in Sec. 6.

2 SYSTEM MODEL AND POWER-PERFORMANCE TRADEOFF

In this section, we first give a brief analysis for a real world mobile app usage trace [20] to show the dynamics and unpredictabilities of mobile users' job requests. Then we give the architecture of the mobile system with C-RAN and MEC, as shown in Fig. 1. After that, we present the dynamic scheduling and model the power-performance tradeoff into a stochastic optimization problem.

2.1 Real World Mobile Trace Analysis

Due to the mobility of mobile devices, mobile users' job requests are always dynamic and unpredictable. Here we take a real world mobile app usage trace from Livelab dataset [20] to show this. The trace contains about 1.4×10^6 job requests from 34 users spanning about 13 months. We first randomly select six users and

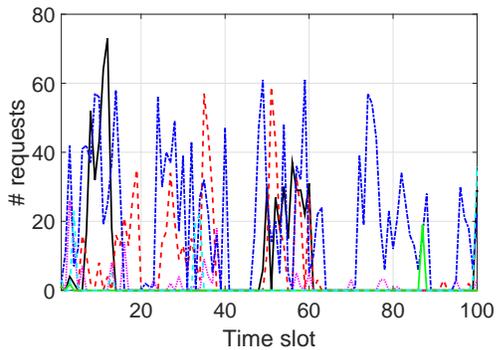


Fig. 2: The arrival of job requests for six sample users from [20].

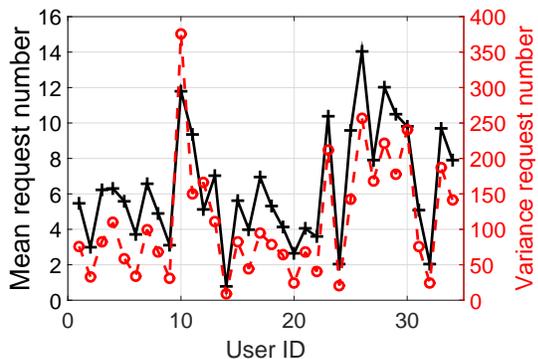


Fig. 3: The mean and variance of number of job requests over time in [20].

plot the arrival of job requests in Fig. 2. Then, we extract the number of job requests for each user over time and plot the *mean* and *variance* of request number in Fig. 3. As shown in the figure, the number of requests from different users have different mean and variance values. The fluctuating and high variance values indicate that the job requests from mobile users in practice are highly dynamic and unpredictable. Such kind of mobility feature introduces huge challenges to the scheduling of computation and network resources for the MSP [16].

2.2 System Architecture

As shown in Fig. 1, the system architecture includes two parts, i.e., C-RAN and edge cloud. There are M RRHs distributed in different geographic locations, and each RRH i serves and receives job requests from a set of mobile users that are close to this RRH. Such a set of users is denoted as a *representative* user set U_i [21]. Accordingly, the mobile system has M sets of users $\mathcal{U} \triangleq \{1, 2, \dots, M\}$. In this paper, a discrete time slotted system has been applied [22], in which the length of a time slot can be several milliseconds or minutes. In every time slot t , $t = 0, 1, 2, \dots$, we model the job requests received by RRH i at time slot t as $(Type_{ij}(t), Size_{ij}(t))$ where $Type_{ij}(t)$ is the job type and $Size_{ij}(t)$ is the input data size. By utilizing the approaches provided in [23], we can obtain the total number of the CPU cycles to be accomplished for each job. Then, we can obtain the running time of each job on a container with fixed resource configuration. After that, we can model job requests for RRH i at time t as $(A_i(t), w_i)$, where $A_i(t)$ denotes the number of job requests with a time average rate $\lambda_i = \mathbb{E}\{A_i(t)\}$. $w_i \in [w^{min}, w^{max}]$ denotes the number of time slots needed for

a job received by RRH i and can be referred to as the *workload* of the job.

Similar to previous work in mobile networking [24], we consider a quasi-static scenario where mobile devices remain unchanged during a time slot. Hence, we can assume that mobile users served by one RRH will not influence other users served by another RRH, without loss of generality. Then over time slots, each variable $A_i(t)$ is independent and identically distributed. Without loss of generality, we use A_i^{max} to denote the maximum of job requests $A_i(t)$. Thus, we have $A_i(t) \leq A_i^{max}, \forall i \in \mathcal{U}, \forall t$. As analyzed in Sec. 2.1, mobile job requests are dynamic and unpredictable. Hence, no priori knowledge of $A_i(t)$ has been assumed in this paper.

The RRHs are connected to the BBU pool via a fronthaul network which consumes power to transmit requests. Dai and Yu's work [25] simply assumes the fronthaul consumption is the accumulated data rates of all users served by RRH and model the fronthaul capability as

$$\bar{C} \leq \bar{C}^{max} \quad (1)$$

then, for a time slot, the i -th fronthaul constraint can be modeled as the maximum number of requests, i.e., $C_i \leq C_i^{max}, \forall i \in \mathcal{U}$.

In the BBU pool, we implement one of the BBUs as a Dispatcher¹ which can receive requests from fronthaul links and route them across several servers in the edge cloud located with the BBU-pool.

Edge cloud consists of N servers $\mathcal{S} \triangleq \{1, 2, \dots, N\}$. Each server $j, \forall j \in \mathcal{S}$ creates containers² which can process the job requests transmitted from the Dispatcher in the BBU pool. We assume that each server creates a container i to process requests from U_i , i.e., container i on server j only serves requests from U_i . This is reasonable because different users have different requirements of hardware/software resources [26]. So container i could misfit other users. In addition, the system can start a container for other users if needed within 1 second [26]. Given a fixed length of each time slot, a container can process a fixed number of job requests. Equivalently, a time slot for computation can be viewed as the process capacity of a container during each time slot. Note that the BBU pool also has many other jobs to do in the C-RAN system (e.g., baseband signal processing, precoding matrix calculation, channel state information estimation [2]), but this is beyond the scope of our paper.

The key notations have been summarized in Table 1.

2.3 Dynamic Scheduling

Fronthaul Scheduling: In every time slot t , $t = 0, 1, \dots$, the mobile system needs to transmit a subset of each user's job requests $R_i(t)$ into the BBU pool through the fronthaul links:

$$0 \leq R_i(t) \leq A_i(t) \quad (2)$$

The fronthaul scheduling policy is to schedule each fronthaul link in time slot t , by tuning to *active* state for transmitting requests from the RRH i to the BBU pool and *sleep* state to decline the requests from mobile users' devices. In C-RAN, the RRH only receives signals from mobile users and then transfers

¹In the future, the Dispatcher can be implemented as a Controller or Manager which can allocate both bandwidth and computational resources.

²Since edge cloud needs to be close to mobile users, it has limited resources compared with remote public clouds while container technology can be more effective than servers with *virtual machines* (VMs) [26].

TABLE 1: Key Notations

Notation	Description
M	number of users
N	number of servers
\mathcal{U}	all user sets, including $U_i, 1 \leq i \leq M$
\mathcal{S}	all servers in edge cloud, including $S_j, 1 \leq j \leq N$
$A_i(t)$	arrival job requests for RRH i at time slot t
w_i	the number of time slots for user i with maximum w^{max}
λ_i	time average rate of $A_i(t)$ with maximum A_i^{max}
C_i	fronthaul capability of fronthaul link i with maximum C_i^{max}
$a_i(t)$	fronthaul scheduling policy
$R_i(t)$	requests transmitted by fronthaul link i at time slot t
$D_{ij}(t)$	job requests dispatched from user i to server j
$X_i(t)$	queue backlog of buffer queue for users i
$b_{ij}(t)$	container scheduling policy in edge cloud
$b_{ij}(t)^-$	left-over container scheduling policy in edge cloud
$Q_{ij}(t)$	queue backlog of each container i in each server j
r_i	time average throughput
a_i	time average transmission capacity for each fronthaul link i
b_{ij}	time average consumed capacity for each container i in server j
p_i^f	time average power consumption of fronthaul link i
p_j^s	time average power consumption of server j
$C_i^l(t)$	container set with left-over jobs running at time slot t on server j
$C_i^s(t)$	container set can finish the job from user i in n -th time interval
$C_i^u n(t)$	container set cannot finish the job from user i in n -th time interval
T	time interval in <i>VariedLen</i>
V	control parameter in Lyapunov technique
α_i	non-negative normalized parameter for U_i
μ	fronthaul link state
ϕ	normalized CPU speed
β	non-negative normalized parameter for fronthaul link
γ	non-negative normalized parameter for edge cloud
$1 - \eta$	normalized power consumption of an idle server

to the BBU pool [2]. The RRH would not undertake computation tasks and cannot buffer job requests. Therefore, for the denied requests, the system can send negative response signals back to the corresponding mobile user. Mobile users can re-send their requests [22] or execute these requests locally [27]. If mobile users still choose to re-send the requests, it means that the local mobile devices are very short of resources. In this way, mobile users are either willing to pay a high price or bear a long latency. Such fronthaul scheduling policies $a_i(t)$ are denoted as the l_0 -norm of $R_i(t)$ (i.e., $a_i(t) = \|R_i(t)\|_0$), which can be indicated by the following function for $\forall i \in \mathcal{U}, \forall t$:

$$a_i(t) = \|R_i(t)\|_0 = \begin{cases} 1 & \text{fronthaul link } i \text{ is on } \textit{active} \text{ state} \\ 0 & \text{fronthaul link } i \text{ is on } \textit{sleep} \text{ state} \end{cases}$$

As mentioned in Sec. 2.2, C_i^{max} refers to as the capacity constraint of each fronthaul link i . Hence, the transmitted requests $R_i(t)$ also need to satisfy the following constraint:

$$R_i(t) \leq a_i(t)C_i^{max}, \forall i \in \mathcal{U}, \forall t \quad (3)$$

Obviously, more requests, i.e., high performance, can be transmitted to the system when turning more fronthaul links to *active* state. But a larger amount of power will be consumed by the fronthauls. Such a tradeoff between power and performance will be characterized in the Sec. 3.1.1.

BBU-based Requests Dispatching: After the subset of requests of each user $R_i(t), \forall i \in \mathcal{U}$ are transmitted to the BBU pool, the Dispatcher in the BBU pool will route those requests to the corresponding container hosted in the edge cloud. We assume that the amount of admitted requests $R_i(t)$ are queued in the buffer for each user set i in the BBU pool before dispatching to the corresponding queue for each container in the edge cloud. Let $X_i(t)$ denotes the backlog of this buffer queue i at time slot t . Also let $D_{ij}(t), \forall i \in \mathcal{U}, \forall j \in \mathcal{S}, \forall t$ denotes the requests dispatched

from user i to server j . We have the following queuing dynamics [18] for the backlog $X_i(t)$,

$$X_i(t+1) = \max\{X_i(t) - \sum_{j=1}^N D_{ij}(t), 0\} + R_i(t) \quad (4)$$

Initially, $X_i(0) = 0, \forall i \in \mathcal{U}$. For each user i , at most $X_i(t)$ requests can be dispatched to servers in the edge cloud. Hence, the dispatching decisions $D_{ij}(t)$ should satisfy the following constraint:

$$\sum_{j=1}^N D_{ij}(t) \leq X_i(t), \forall i \in \mathcal{U} \quad (5)$$

Cloud Server Scheduling: After dispatching requests $D_{ij}(t)$ to the corresponding container i on each server j , the last scheduling policy is to schedule each container in time slot t , by stopping the container to *shutdown* state to keep the requests waiting in this container's queue, without processing them in the current time slot, or starting the container to *running* state to process the dispatched user requests that are waiting in this container's queue. Note that the system will allocate edge servers once they are available and mainly consider the scalability [28] of container in this paper. The reason is that mobile users are more sensitive to the latency and can not wait for the long rebooting time for a server.

Once the container i on server j has started to *running* state to process the job request from user i , the job will occupy this container for different time slots based on the length of the job request. Hence, the scheduling policy $b_{ij}(t)$ is to schedule each container in time slot t by starting the container to *running* or stopping to *shutdown* state. However, since job requests with varied lengths need to be executed in consecutive time slots, we introduce $b_{ij}(t)^-$ to denote the job scheduled before t , which is still running on container i on server j in t . That is to say, $b_{ij}(t)^- = 1$ means that a left-over job is running on this container. Once container i is scheduled to serve a job from user i , the job will run for w_i consecutive time slots, i.e., $b_{ij}(t+k)^- = 1, k = 1, \dots, w_i - 1$. At the same time, the container i will be *running* state for the following $w_i - 1$ time slots and cannot serve other jobs for user i , i.e., $b_{ij}(t+k) = 0, k = 1, \dots, w_i - 1$. The server scheduling policies for $\forall i \in \mathcal{U}, \forall j \in \mathcal{S}, \forall t$ can be given as the following indicator function:

$$b_{ij}(t) = \begin{cases} 1 & \text{container is } \textit{running} \text{ state and } b_{ij}(t)^- = 0 \\ 0 & \text{container is } \textit{shutdown} \text{ state or } b_{ij}(t)^- = 1 \end{cases}$$

Accordingly, we can derive the queue backlog, arrival rate and service rate of each container as follows. We assume the servers from the edge cloud are homogeneous¹. In each server, we create the same container for each user. At every time slot t , the container can process one job request from mobile users. Let $Q_{ij}(t)$ denotes the total unprocessed workloads of container i on server j . At the beginning of time slot t , $Q_{ij}(t)$ workloads are waiting in the queue with $Q_{ij}(0) = 0$. The service rate of $Q_{ij}(t)$ can be quantified as $b_{ij}(t) + b_{ij}(t)^-$, where $b_{ij}(t)$ denotes the newly scheduled job from user i at the beginning of time slot t and $b_{ij}(t)^-$ denotes the unfinished job (or left-over job) for user i . The arrival rate is $w_i D_{ij}(t)$.

¹In the future, we can easily extend the edge cloud to the heterogeneous environment by considering more complicated model.

Apparently, we can have the following queuing dynamics over time for each container hosted in each server as follows:

$$Q_{ij}(t+1) = \max\{Q_{ij}(t) - b_{ij}(t) - b_{ij}(t)^-, 0\} + w_i D_{ij}(t) \quad (6)$$

Obviously, more job requests (high performance) can be processed when starting more containers (i.e., on the *running* state) in edge cloud. But a larger amount of power will be consumed by servers. Such a tradeoff needs to characterize in the following subsection.

Now we have modeled the dynamic scheduling for both network resources in C-RAN and computation resources in MEC. Then, we will propose the tradeoff between power consumption and performance in the next section.

2.4 Power-Performance Tradeoff

2.4.1 Time Average Throughput

In the mobile system, the overall system throughput (i.e., the processing jobs) is one of the most significant performance metrics. Especially, we define the time average throughput r_i for each user U_i as follows,

$$r_i = \lim_{t \rightarrow \infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \mathbb{E}\{w_i R_i(\tau)\}, \forall i \in \mathcal{U} \quad (7)$$

Together with r_i , we define the time average transmission capacity a_i for each fronthaul link i :

$$a_i = \lim_{t \rightarrow \infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \mathbb{E}\{a_i(\tau)\}, \forall i \in \mathcal{U} \quad (8)$$

Then we can define the time average consumed capacity b_{ij} for each container i on server j :

$$b_{ij} = \lim_{t \rightarrow \infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \mathbb{E}\{b_{ij}(\tau) + b_{ij}(\tau)^-\}, \forall i \in \mathcal{U}, \forall j \in \mathcal{S} \quad (9)$$

Then the overall MEC throughput is $\sum_{i=1}^M r_i$, which is constrained as follows: (1) $r_i/w_i \leq \lambda_i$, i.e., the time average throughput cannot exceed the time average arrival rate for any mobile user i ; (2) $r_i/w_i \leq a_i C_i^{max}$, as the time average throughput cannot exceed the capacity of the fronthaul link between RRH i and the BBU pool; (3) $r_i/w_i \leq \sum_{j=1}^N b_{ij}$, as the time average throughput r_i cannot exceed the overall processing capacity allocated for user i , $\forall i \in \mathcal{U}, \forall j \in \mathcal{S}$.

2.4.2 Time Average Power Consumption

We analyze two power consumption models in this part. The first one is the power consumption of the fronthaul links connecting each RRH with the BBU pool, as fronthaul capacity is one of the most important limitations in C-RAN [2]. The other one is servers' power consumption in the edge cloud which process all job requests.

For the power of fronthaul, it consumes a constant power when it is on *active* state [25], [29]. Without loss of generality, we consider a normalized power consumption $P^f(\mu) \in \{0, 1\}$, where $\mu = 0$ represents the *sleep* state of a fronthaul link while $\mu_1 = 1$ represents the *active* state:

$$P^f(\mu) = \mu \quad (10)$$

Based on the fronthaul power model above, for a fronthaul link $i \in \mathcal{U}$ that transmits the requests from the RRH to the BBU

pool with the scheduling policy $a_i(t)$ described in Sec. 2.3, its normalized power consumption in time slot t is given as follows:

$$P_i^f(t) = P_i^f(a_i(t)) = a_i(t) \quad (11)$$

Accordingly, for each fronthaul link $\forall i \in \mathcal{U}$ in the C-RAN, we have the normalized power consumption p_i^f as follows,

$$p_i^f = \lim_{t \rightarrow \infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \mathbb{E}\{P_i^f(\tau)\} \quad (12)$$

then, $\sum_{i=1}^M p_i^f$ is the overall time average power consumption of all fronthaul links.

For the power of server, it has been widely studied [30], [31] that a server's power consumption is principally related to the running CPU speed ϕ . So we follow this fact and ignore the other resource of power consumption in the servers (e.g., memory and network) and employ a very basic server power model to characterize the normalized power consumption as follows,

$$P^s(\phi) = \eta \phi^v + (1 - \eta) \quad (13)$$

Without loss of generality, a normalized speed $0 \leq \phi \leq 1$ and its corresponding normalized power consumption $P^s(\phi)$ are considered in this paper. Intuitively, the container stops to *shutdown* state when $\phi = 0$ and starts to *running* state with maximum CPU speed $\phi = 1$. Hence, the normalized CPU speed of server j for the cloud server scheduling model can be given as $\phi_j(t) = \sum_{i=1}^M \frac{b_{ij}(t) + b_{ij}(t)^-}{M}$. For parameter v , we set it empirically as $v > 1$ in practical [31]. With another parameter $0 \leq \eta \leq 1$, we denote $1 - \eta$ as an idle server's power consumption. In this paper, we will schedule the container between *running* and *shutdown* state while the container with *shutdown* state will not consume computation resources (e.g., CPU). This is different from our previous version [32] that switches VMs between running and idle state. The main difference is that the startup of a container is very fast [17] but VM requires considerable startup overhead [33]. Hence, we can stop the container to *shutdown* state without processing requests and start the container to *running* state very fast. But if we shut down a VM, it needs significant time to boot-up again. It is worth noting that it also needs a lot of time to start a server. Hence, we will not power off a server even when it is idle.

Based on the above power consumption model, the power consumption for server j are given as follows,

$$P_j^s(t) = \eta (\phi_j(t))^v + (1 - \eta) \quad (14)$$

Accordingly, the time average of normalized power consumption of each server j in the mobile system can be defined as follows,

$$p_j^s = \lim_{t \rightarrow \infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \mathbb{E}\{P_j^s(\tau)\} \quad (15)$$

then the overall servers' power consumption in the edge cloud is $\sum_{j=1}^N p_j^s$.

For the MSP, power for both fronthaul links and servers is hopefully being minimized as these power incurs a large amount of electricity cost.

2.4.3 Time Average Profit Maximization

Now, we have obtained the MEC throughput metric r_i in Eq. (7), the fronthaul power consumption metric p_i^f in Eq. (12) and the server power consumption metric p_j^s in Eq. (15). We define our scheduling objective as the MSP's time average profit as follows:

Time average throughput revenue: MSP' revenue gained from all mobile users can be effected by multiple factors, e.g., the throughput and the usage of data for transfer job requests. We measure MSP's time average revenue gained from all mobile users as

$$\bar{e}_t = \sum_{i=1}^M \alpha_i r_i + \sum_{i=1}^M d_i \quad (16)$$

where α_i is a non-negative normalized parameter for each U_i . The parameter α_i allows us to reply to different scenarios. For example, we can set the values the same for all users if we treat them equally. Also, we can assign priorities [34] for different mobile users by choosing appropriate values of α_i . In this situation, we can assign high priority to mobile users who have to offload their requests to edge cloud due to the limited computing resource. While for mobile devices with powerful capacity, low priority can be assigned. Moreover, we can design dynamic pricing [21] for the requests as some mobile users are willing to pay more to process their job requests in MEC. d_i denotes the cost of the data usage goody-bag, e.g., 1 Gigabytes per month, purchased from MSP for each user i [35]. Other factors such as bandwidth between users and the mobile system may also affect MSP' profit. But from the perspective of mobile users, they can not decide the bandwidth. Thus, they are probably not willing to pay for the bandwidth. To make our model traceable, we assume the revenue is related to the throughput and the usage of data, but without much loss of generality.

Time average fronthaul electricity cost: The electricity cost of fronthaul power in C-RAN can be measured as $\bar{e}_f = \sum_{i=1}^M \beta p_i^f$, where $\beta = Price_f \times PUE_f$ is a non-negative normalized parameter. $Price_f$ is the electricity market price of each unit of the normalized power consumption of fronthaul. PUE_f is the the *power usage efficiency* (PUE) defined as the ratio of the total facility power used by the entire equipment to the actual power consumed for the IT equipment (fronthaul) [22].

Time average server electricity cost: Similarly, the electricity cost of server power can be measured as $\bar{e}_s = \sum_{j=1}^N \gamma p_j^s$, where $\gamma = Price_s \times PUE_s$ is a parameter for the server power. $Price_s$ is the electricity market price of each unit of the normalized power consumption of server while PUE_s is the PUE for servers j in edge cloud. We assume the values of PUE_s for all servers are the same as they are managed by the same MSP.

Given the above time average revenue brought by the throughput and cost for both fronthaul and server power, we formulate the maximization of time average profit as the following *stochastic optimization* problem:

$$\mathcal{P} : \quad \max_{\alpha_i, D_{ij}, b_{ij}} \quad \bar{e}_t - \bar{e}_f - \bar{e}_s \quad (17)$$

$$s.t. \quad (2), (3), (5)$$

It is very difficult to solve the above Problem (\mathcal{P}) in an offline and centralized manner. This is because it requires offline future information about mobile users' requests. Since mobile users always follow their own mobility [16] and the arrival of job requests are unpredictable, we cannot get the future information

about mobile users' requests. Meanwhile, it suffers from "the curse of dimensionality" and is computationally intractable when the problem scales up. Our considerations on power consumption and queue stability lead us to design online resource scheduling algorithms based on the Lyapunov optimization framework [18], which has been widely used in power consumption optimization problem [19], [22].

3 ONLINE ALGORITHM FOR JOBS WITH VARIED LENGTHS

In this section, to address the challenges of optimization Problem (\mathcal{P}), we take advantage of the Lyapunov optimization technique [18] to design a resource online scheduling algorithm called *VariedLen*. Our *VariedLen* algorithm designs scheduling policies including the fronthaul scheduling, requests dispatching and server scheduling simultaneously. It can be proved that the algorithm can achieve a time average profit that is close to the optimum of Problem (\mathcal{P}).

3.1 Problem Transformation Using Lyapunov Optimization

3.1.1 Characterizing the Stability-Profit Tradeoff

Denoting $\mathbf{Q}(t) = (Q_{ij}(t))$ and $\mathbf{X}(t) = (X_i(t))$ as the matrixes of queues maintained by containers in the edge cloud and the buffer queues for mobile users in the BBU pool. After that, we use $\Theta(t) = [\mathbf{Q}(t); \mathbf{X}(t)]$ to represent the combined matrix of queues. Since $\mathbf{X}(t)$ and $\mathbf{Q}(t)$ have different scales ($\mathbf{X}(t)$ corresponds to the number of job requests (Eq. 4), while $\mathbf{Q}(t)$ corresponds to the request length w_i (Eq. 6)), we assign queue $X_i(t)$ and $Q_{ij}(t)$ with different weights w_i and 1 and have the Lyapunov function $L(\Theta(t))$ as Eq. (18).

$$L(\Theta(t)) = \frac{1}{2} \left\{ \sum_{i \in \mathcal{U}} w_i^2 X_i^2(t) + \sum_{i \in \mathcal{U}} \sum_{j \in \mathcal{S}} Q_{ij}^2(t) \right\} \quad (18)$$

This function is a scalar metric of congestion [18] for the edge cloud. Intuitively, all queue backlogs are small when $L(\Theta)$ is small. That is, the corresponding mobile system has strong stability. Based on Eq. (18), we define the conditional *1-slot Lyapunov drift* [18] as follows:

$$\Delta(\Theta(t)) = \mathbb{E}\{L(\Theta(t+1)) - L(\Theta(t)) | \Theta(t)\} \quad (19)$$

Under the Lyapunov optimization, the scheduling policies $a_i(t)$, $D_{ij}(t)$ and $b_{ij}(t)$ should be chosen to minimize the infimum bound on the following drift-minus-profit [18] in every time slot t :

$$\Delta(\Theta(t)) - V \mathbb{E} \left\{ \sum_{i \in \mathcal{U}} \alpha_i w_i R_i(t) + \sum_{i \in \mathcal{U}} d_i - \beta \sum_{i \in \mathcal{U}} P_i^f(t) - \gamma \sum_{j \in \mathcal{S}} P_j^s(t) \mid \Theta(t) \right\} \quad (20)$$

The parameter $V \geq 0$ is used to balance *the tradeoff between the profit maximization and the drift*. For example, a high value of V indicates that the mobile system prefers to achieve more profit rather than keep the system queue backlogs at a low level.

3.1.2 Bounding the Drift-Minus-Profit

We need the following Lemma to derive the infimum bound of the drift-minus-profit given in Eq. (20),

Lemma 1. Given any scheduling policies at any time slots, the following inequality for drift-minus-profit (Eq. 20) can be derived:

$$\begin{aligned} \Delta(\Theta(t)) - V\mathbb{E}\left\{\sum_{i=1}^M \alpha_i w_i R_i(t) + \sum_{i=1}^M d_i - \beta \sum_{i=1}^M P_i^f(t) - \gamma \sum_{j=1}^N P_j^s(t) \mid \Theta(t)\right\} \leq B \\ - \sum_{i=1}^M \mathbb{E}\{R_i(t)(V\alpha_i w_i - X_i(t)w_i^2) - V\beta a_i(t) \mid \Theta(t)\} \quad (21) \end{aligned}$$

$$- \sum_{i=1}^M \sum_{j=1}^N \mathbb{E}\{D_{ij}(t)(w_i^2 X_i(t) - w_i Q_{ij}(t)) \mid \Theta(t)\} \quad (22)$$

$$- \sum_{j=1}^N \mathbb{E}\left\{\sum_{i=1}^M Q_{ij}(t)(b_{ij}(t) + b_{ij}(t^-)) - V\gamma P_j^s(t) \mid \Theta(t)\right\} \quad (23)$$

where $B = \frac{[MN+3\sum_{i=1}^M(\max\{A_i^{max}, C_i^{max}\})^2]}{2} - V\sum_{i=1}^M d_i$.

Proof: See Appendix A. \square

Now we have transformed the stochastic optimization problem (\mathcal{P}) into the bounding of *Drift-Minus-Profit* by using Lyapunov optimization. By minimizing the infimum bound in Lemma 1, we can design an optimal resource online scheduling algorithm in the next section.

A highlight of this paper is that previous work using standard Lyapunov optimization [18], [22], [36] usually assume each job request can be completed in one time slot, while we model a more general scenario and allow a job request with length longer than a time slot. In this way, job requests can not be prematurely terminated once scheduled to run on the containers. This constrains the scheduling decisions in the current time slot.

3.2 Optimal Resource Online Scheduling Algorithm (VariedLen)

In this subsection, *VariedLen* has been proposed to minimize the infimum bound in Lemma 1 by equivalently maximizing the terms (21) (22) (23) on the right-hand-side (RHS). In each time slot t , $t = 0, 1, 2, \dots$, our *VariedLen* schedules resources in MEC and C-RAN by maximizing the terms (21) (22) (23), including fronthaul scheduling Problem ($\mathcal{P}1.1$) in Sec. 3.2.1, requests dispatching Problem ($\mathcal{P}1.2$) in Sec. 3.2.2 and server scheduling Problem ($\mathcal{P}1.3$) in Sec. 3.2.3. After that, we update all queues by using Eq. (4) and Eq. (6).

3.2.1 Fronthaul Scheduling

In this subsection, we will solve the first challenge in the resources scheduling, i.e., how to schedule each fronthaul link. For every mobile user set $U_i, i \in \mathcal{U}$ shown in Fig. 1, we can maximize the term (21) in Lemma 1 to derive the fronthaul scheduling policies $a_i(t), i = 1, 2, \dots M$. Recall that different mobile devices served by different RRHs cannot influence each other in our system (see Sec. 2). Therefore, the fronthaul scheduling policy $a_i(t)$ for different U_i are independent which means that the maximization

of (21) can be decomposed to compute the following Problem ($\mathcal{P}1.1$) concurrently.

$$\begin{aligned} \mathcal{P}1.1 : \max_{a_i(t)} R_i(t)(Vw_i\alpha_i - w_i^2 X_i(t)) - V\beta a_i(t) \quad (24) \\ s.t. \quad (2), (3) \end{aligned}$$

Problem ($\mathcal{P}1.1$) includes two parts, the first one $R_i(t)(Vw_i\alpha_i - w_i^2 X_i(t))$ is a simple linear programming problem. But the second one $V\beta a_i(t)$ (i.e., $V\beta \|R_i(t)\|_0$) is a l_0 -norm problem which is hard to solve. However, inspired by compressive sensing, l_1 -norm is the best convex relaxation of the l_0 -norm since l_1 -norm is the convex envelop of l_0 -norm [37], [38]. By applying l_1 -norm relaxation to the Problem ($\mathcal{P}1.1$) and rearranging the terms, we have the following relaxed problem,

$$\begin{aligned} \max_{a_i(t)} R_i(t)(Vw_i\alpha_i - V\beta - w_i^2 X_i(t)) \quad (25) \\ s.t. \quad (2), (3) \end{aligned}$$

The above problem is a simple linear programming problem and we can derive the optimal value of $R_i(t)$ as:

$$R_i(t) = \begin{cases} \min\{A_i(t), C_i^{max}\}, & X_i(t) < \frac{Vw_i\alpha_i - V\beta}{w_i^2} \\ 0, & \text{else} \end{cases} \quad (26)$$

then we can have the fronthaul scheduling policies for Problem ($\mathcal{P}1.1$) as:

$$a_i(t) = \|R_i(t)\|_0 = \begin{cases} 1, & X_i(t) < \frac{Vw_i\alpha_i - V\beta}{w_i^2} \\ 0, & \text{else} \end{cases} \quad (27)$$

The optimal solution of Problem ($\mathcal{P}1.1$) is a simple threshold-based scheduling policy. When the backlog $X_i(t)$ of the buffer queue for U_i is smaller than a threshold $X_i(t) < \frac{Vw_i\alpha_i - V\beta}{w_i^2}$, the fronthaul will transmit as many job requests (newly received by RRH i) as possible, but it cannot exceed the capacity limitation of the fronthaul link i . When $X_i(t)$ is higher than the threshold, it means that the system is overloaded and it will decline all the requests to make the system stable. The intuition of this policy is two-fold: when the backlog of the buffer queue for U_i is smaller than the threshold $X_i(t) < \frac{Vw_i\alpha_i - V\beta}{w_i^2}$, then MEC throughput increases by transmitting as many requests as possible into the BBU pool which can improve the profit. On the other hand, when the backlog of the buffer queue is larger than the threshold, the system will decline all the requests to make the mobile system stable. By doing so, the scheduling policy can prevent the BBU pool with reasonable backlogged requests from being overloaded by newly received requests.

If one wants to focus on the latency, the system can tune a high value of V to increase the value of threshold. At the same time, more containers will be scheduled to running state for requests processing according to the greedy policy for cloud servers (see Section 3.2.3 for details). In this way, the system can guarantee the latency requirement of each request.

3.2.2 BBU-based Requests Dispatching

While in this part, we will solve the job requests dispatching challenge for different mobile users. For each $U_i, i \in \mathcal{U}$ shown in Fig. 1, we can maximize the term (22) in Lemma 1 to derive the BBU-based dispatching policies $D_{ij}(t), j = 1, 2, \dots N$. Similar to the fronthaul scheduling policies, mobile users are independent from each other. Therefore, the requests dispatching policies $D_{ij}(t)$ of different U_i are also independent which means

that the maximization of (22) can be decomposed to compute the following Problem (P1.2) concurrently.

$$\begin{aligned} \mathcal{P1.2} : \max_{D_{ij}(t)} & \sum_{j=1}^N D_{ij}(t)(w_i^2 X_i(t) - w_i Q_{ij}(t)) \\ \text{s.t.} & \end{aligned} \quad (28)$$

The above Problem (P1.2) is a weighted linear programming problem, in which the dispatched requests to server j for U_i 's buffer queue is weighted by $w_i^2 X_i(t) - w_i Q_{ij}(t)$. Note that for each U_i at time slot t , the value $X_i(t)$ is constant. Hence, the optimal dispatching strategy for each U_i tends to dispatch as many buffered request as possible to the container with the least backlog:

$$D_{ij}(t) = \begin{cases} X_i(t) & j = j_i(t) \text{ and } X_i(t) > \frac{Q_{ij_i(t)}(t)}{w_i} \\ 0 & \text{otherwise} \end{cases} \quad (29)$$

where $j_i(t) = \arg \min_{j \in \mathcal{S}} Q_{ij}(t)$ means the queue with the shortest backlog in all N queues on N containers for U_i . Such a dispatching policy is accord with the join-the-shortest-queue (JSQ) policy for load balancing in cluster computing [39]. The intuition of JSQ policy is to reduce the response delay of newly received requests by preferentially dispatching to the shortest queue.

3.2.3 Mobile Server Scheduling

In this subsection, we will address the challenge about how to schedule all containers hosted on each server j for each time slot t . The *running* or *shutdown* state of each container on server j can be scheduled by maximizing the term (23) in Lemma 1. Recall that the power consumption model is based on the individual server in Sec. 2.4.2, therefore the indicator function $b_{ij}(t)$ are independent among different servers. The maximization of term (23) can be decomposed into the following subproblem (P1.3) for every individual server j :

$$\begin{aligned} \mathcal{P1.3} : \max_{b_{ij}(t)} & \sum_{i=1}^M Q_{ij}(t)(b_{ij}(t) + b_{ij}(t)^-) - V\gamma P_j^s(t) \\ \text{s.t.} & \end{aligned} \quad (30)$$

The above Problem (P1.3) can be solved by using enumeration method, i.e., switching all the possible combination of containers to *running* state and searching the maximized value of $\sum_{i=1}^M Q_{ij}(t)(b_{ij}(t) + b_{ij}(t)^-) - V\gamma P_j^s(t)$ in Problem (P1.3). But the exponential complexity is impracticable when the containers in edge cloud scale up to hundreds and thousands. Hence, we seek to design a greedy strategy as follow.

For Problem (P1.3), we not only have to schedule the containers on server j at time slot t , but also consider the container with running left-over jobs before time slot t . Hence, we first need to distinguish the container whether it has left-over job running or not on server j . We denote the subset containers of all M containers on server j with left-over jobs running at time slot t as $C_j^l(t) = \{k | b_{kj}(t)^- = 1, 1 \leq k \leq M\}$. For those containers in $C_j^l(t)$, we have $b_{ij}(t) = 0$ according to the scheduling policy of $b_{ij}(t)$ in Sec. 2.3. For Problem (P1.3), we need to schedule the containers that are not in C_j^l . Then we can change Problem (P1.3) into the following problem,

$$\begin{aligned} \max_{b_{ij}(t)} & G_1 + \sum_{i \notin C_j^l(t)} Q_{ij}(t)b_{ij}(t) \\ & - V\gamma\eta(G_2 + \frac{\sum_{i \notin C_j^l} b_{ij}(t)}{M})^v \\ \text{s.t.} & \end{aligned} \quad (31)$$

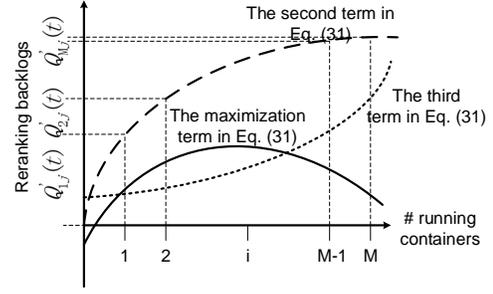


Fig. 4: The illustration of optimal solutions for Problem (P1.3), which reflects the shape of the second term, the third term and the maximization term in Eq. (31).

where $G_1 = \sum_{i \in C_j^l(t)} Q_{ij}(t) + 1 - \eta$ is a constant for a specific server j at time slot t . $G_2 = 1 - \frac{M'}{M}$ is also a constant with $M' = |C_j^l|$.

Intuitively, the solution of the above problem remains the same if we remove the constant G_1 . At the same time, we find that the scheduling policy $b_{ij}(t)$ in server j is weighted by the queue backlog $Q_{ij}(t)$ of container i while the power consumption growth incurred by starting each container is the same under our server power consumption model (recall the model of $P_j^s(t) = \eta(\sum_{i=1}^M b_{ij}(t)/M)^v + (1-\eta)$ in Sec. 2.4.2). Therefore, if we rank all containers hosted on server j according to their queue backlog in descending order (i.e., $Q_{1j}(t)' \geq Q_{2j}(t)' \geq \dots \geq Q_{Mj}(t)'$), then Fig. 4 can illustrate the optimal solution of Problem (P1.3). One can search from the container with the most backlog (i.e., $Q_{1j}(t)'$) to the container with the least backlog (i.e., $Q_{Mj}(t)'$). First, if the container is running with a left-over job, then this container is *running* state now and can not schedule to run a new job request, i.e., $b_{ij}(t) = 0$. If the container is *shutdown* state now, we then check whether the growth of the second term $\sum_{i \notin C_j^l(t)} Q_{ij}(t)b_{ij}(t)$ exceeds the power consumption growth (i.e., the third term $V\gamma\eta(G_2 + \frac{\sum_{i \notin C_j^l} b_{ij}(t)}{M})^v$) incurred by starting a container i or not. If it is true, container i needs to schedule to the *running* state. Once the growth of the second term is smaller than the growth of the third term for container i , we need to schedule container i and the other containers to the *shutdown* state.

The above solutions for Problem (P1.1-P1.3) can make decisions on fronthaul link scheduling, BBU-based dispatching and the server scheduling in the edge cloud at every time slot. In the standard Lyapunov optimization framework, as used in many previous studies [22], [32], it critically assumes that all job requests have fixed length equivalent to the length of a time slot. However, in this paper, we consider a more realistic and general scenario in which mobile jobs have variable lengths denoted as w_i time slots in our model. The cloud server scheduling decisions made in time slot t directly affect the server scheduling in later time slots (i.e., time slot $t+1, \dots, t+w_i-1$). Such kind of decisions in consecutive time slots is beyond what standard Lyapunov technique can handle. Hence we need to design a new resource online scheduling algorithm, *VariedLen*, to handle job requests with *varied lengths* as follows.

We first divide the total time slots into several *time intervals* and each time interval I_n has T time slots with $T > w^{max}$. Then, we can make decisions for each time slot in each time interval. Since the fronthaul scheduling and BBU-based request dispatching

have not involved with the consecutive scheduling of container in the edge cloud, the solutions of fronthaul scheduling Eq. (27) and BBU-based request dispatching Eq. (29) remain unchanged. While for the mobile server scheduling, at each time slot t , we divide the containers for different mobile users into two subsets. At time slot t of a time interval I_n , we will start containers to *running* state to run jobs from user i only when the job can be finished in this time interval. We denote these containers as a container set $C_j^s(t)$, i.e., $C_j^s(t) = \{i | nT \leq t \leq (n+1)T - w_i\}$. The other containers will not be scheduled at time slot t and we denote them as a container set $C_j^{un}(t)$, i.e., $C_j^{un}(t) = \{i | (n+1)T - w_i + 1 \leq t \leq (n+1)T - 1\}$. After that, we use the solutions described for Problem (P1.3) to make decisions among the container set $C_j^s(t)$. The detailed algorithm of *VariedLen* has been presented in Alg. 1.

Algorithm 1 *VariedLen*

Input: $V, \alpha_i, \beta, \gamma, \eta, A_i(t), w_i$.
Output: $a_i(t), D_{ij}(t), b_{ij}(t), i \in \mathcal{U}, j \in \mathcal{S}$.
1: Get $X_i(t)$ and $Q_{ij}(t)$ at the beginning of each time slot t .
2: Get the optimal fronthaul scheduling policies $a_i(t)$ as Eq. (27), BBU-based requests dispatching policies $D_{ij}(t)$ as Eq. (29).
3: **for** each server j **do**
4: Get container sets $C_j^s(t)$ and $C_j^{un}(t)$.
5: Set $b_{ij}(t) = 0$ if $i \in C_j^{un}(t)$.
6: **for** containers in $C_j^s(t)$ **do**
7: Use the solution for Problem (P1.3)
8: **end for**
9: **end for**
10: Update $X_i(t)$ and $Q_{ij}(t)$ according to Eq. (4) and Eq. (6), respectively.

Finally, the queues $X_i(t)$ can be updated according to Eq. (4) based on the optimal values of $R_i(t)$ and $D_{ij}(t)$. The queues $Q_{ij}(t)$ for each container in the edge cloud can be updated with Eq. (6) by using the optimal values of $D_{ij}(t)$ and $b_{ij}(t)$.

For a given time slot t , the fronthaul scheduling policies $a_i(t)$ with Eq. (27) cost $O(M)$, and the dispatching policies $D_{ij}(t)$ with Eq. (29) cost $O(MN)$. While for the cloud server scheduling policies $b_{ij}(t)$, it costs at most $O(M \log M)$ to sort M queue backlogs for each server. Thus the time complexity of Alg. 1 is $O(MN \log M)$.

Since the edge cloud is at the edge of network and close to mobile users, the resource is limited and N is small compared with the number of mobile users (i.e., M). Hence, the complexity of algorithm can approximate to $O(M \log M)$ and be implemented in an online way. When the requests rush into a peak load, the edge cloud can use mature cloud scalability techniques, such as autoscaling [28], to increase the processing capacity.

3.3 Optimality Analysis

Theorem 1. For any arrival rate in any time slot $A_i(t) \leq A_i^{max}$, $\forall i \in \mathcal{U}, \forall t$, implementing the *VariedLen* with any $V \geq 0$ satisfies the following performance bounds:

(1) The queue backlog $X_i(t)$ for U_i buffered in the BBU pool and $Q_{ij}(t)$ for U_i on any server j are upper bounded as follows,

$$X_i(t) \leq V\alpha_i + \min\{A_i^{max}, C_i^{max}\} \quad (32)$$

$$Q_{ij}(t) \leq V\alpha_i + 2\min\{A_i^{max}, C_i^{max}\} \quad (33)$$

(2) The time average profit gained by Alg. 1 is close to the optimal value within a gap (B/V) :

$$\liminf_{t \rightarrow \infty} \inf \left\{ \sum_{i=1}^M \alpha_i r_i - \beta \sum_{i=1}^M p_i^f - \gamma \sum_{j=1}^N p_j^s \right\} \geq \eta^* - \frac{B}{V} \quad (34)$$

where $\eta^* = \sum_{i=1}^M \alpha_i r_i^* - \beta \sum_{i=1}^M p_i^{f*} - \gamma \sum_{j=1}^N p_j^{s*}$, and r_i^*, p_i^{f*} and p_j^{s*} are the optimal values of Problem (P) and $B = \frac{[MN+3 \sum_{i=1}^M (\max\{A_i^{max}, C_i^{max}\})^2]}{2} - V \sum_{i=1}^M d_i$.

Proof: See Appendix B. \square

4 EVALUATION

In this section, we evaluate our proposed algorithm, *VariedLen*, by conducting simulations with a real world mobile app access trace from LiveLab dataset [20]. In the following, we first introduce our experimental setup and methodology, then present the experimental results.

4.1 Experimental Setup

LiveLab [20] is a methodology to measure real-world smartphone usage and wireless networks with a reprogrammable in-device logger designed for long-term user studies in Rice University. The dataset includes 34 students with different devices, e.g., phones and tablets, from Rice University and Houston Community College during February 2010 to February 2011. It consists about 1.4×10^6 jobs from variety of mobile apps, e.g., social network services, video and mobile games. Each job request in the dataset has information about the name of the application, the start time and the duration.

Table 2 presents the details of the mobile app usage trace LiveLab [20]. For each mobile user, we extract the number of requests as $A_i(t)$ at each time slot t . The length of each time slot is 1 second. In this way, we can get a maximum request number from all users for all time slots as A_i^{max} . After that, we will conduct simulation with 3.3×10^7 time slots in total.

TABLE 2: Description of LiveLab dataset

Source	Rice University
Time Duration	13 months
# of job requests	1.4×10^6
# of users	34
# of time slots	3.3×10^7

Table 3 shows all parameters used in the following experiments. Due to the limited resources in the edge cloud, we set the server number N as 40 and create a container for each mobile user on each server in the edge cloud. Mobile user can offload requests to at most 40 containers across servers, with a maximum processing capacity of 40. For the workload w_i of each user i , we randomly select a value from $[w^{min}, w^{max}] = [2, 20]$. We set $d_i = 10$ as the same price of China Mobile for 1 Gigabytes per month [2]. We set the parameter $v = 2, \eta = 0.5, \beta = 0.6$ and $\gamma = 2$ empirically in this paper [22], [31]. For the parameter α , we first set them as 1 for all users. For the length of the time interval T , we first set $T = w^{max}$.

TABLE 3: Parameters used in the simulation

M	34	v	2
N	40	η	0.5
w^{min}	2	α_i	$1, \dots, 1$
w^{max}	20	β	0.6
T	20	γ	2

In the following part, we will import the job request information from the LiveLab dataset and implement the *VariedLen* algorithm by using C language with Microsoft Visual C++ 6.0.

4.2 Experimental Methodology

The proposed *VariedLen* algorithm includes three scheduling policies, i.e., *threshold-based* policy for fronthaul links, *JSQ* policy for the BBU dispatching, and *greedy* strategy for cloud server scheduling. We compare these methods to three classic scheduling methods as follows,

- *Best-effort* (B). For the fronthaul links scheduling, we compare the *threshold-based* (T) with this method which transfers job requests as much as possible.
- *Round-robin* (R). For the BBU dispatching, we compare the *JSQ* (J) policy with this classic scheduling method which dispatches job requests to servers in circular order.
- *First-come-first-served, FCFS* (F). For the mobile server scheduling, we compare the *greedy* (G) strategy with this method which runs the job requests waited in the queue one by one.

The above classic scheduling algorithms have been widely used in the literature [40]–[42]. Meanwhile, when utilizing the Lyapunov technique, the state-of-the-art research often derive or compare with the above classic scheduling algorithms [22], [43], [44]. For example, Zhou et al derived the *JSQ* scheduling when using Lyapunov technique in SaaS cloud [22]. Nan et al used Lyapunov technique to design algorithms in Cloud of Things system and compared them with the *round-robin* algorithm [43], [44].

In the following part, we first conduct several experiments to compare *VariedLen* with the mixture of the above methods in Sec. 4.3. Then, we show the effectiveness of scheduling policies in Sec. 4.4.

Since mobile job requests always have varied lengths while standard Lyapunov technique can not handle, we design the *VariedLen* algorithm by extending the standard Lyapunov technique used in our preliminary work [32]. To demonstrate the novelty of this paper, we compare the *VariedLen* algorithm with the *RICH* algorithm proposed in [32] in Sec. 4.5.

At last, we conduct experiments to show the sensitivity of parameter α_i , β , γ and η in time average profit maximization Problem (P) for *VariedLen* algorithm in Sec. 4.6.

4.3 Algorithm Optimality and System Stability

Our proposed *VariedLen* consists three scheduling methods, i.e., the *threshold-based*, the *JSQ* and the *greedy* method shown in Sec. 3.2. For comparison purpose, we compare these methods with three classic scheduling in computing, i.e., *best-effort*, *round-robin* and *FCFS*. We mix these methods and form eight scenarios, from TJG to BRF. For example, BRF means we use *best-effort*, *round-robin* and *FCFS* methods respectively. Obviously, TJG is equivalent to our *VariedLen* algorithm. Fig. 5 shows the time average profit for all scheduling methods under different V , while Fig. 6 shows the congestion.

From Fig. 5 we can see, (1) the time average profit increases and converges to the optimum for larger values of V . This verifies *Theorem 1* in that the profit gained by *VariedLen* is close to the optimal profit captured by Eq. (34) with a diminishing gap ($1/V$). However, with an excessive high of V , the improvement starts to diminish which can aggravate the congestion of queues in the system (captured by Eq. (18)). The profit grows rapidly when $V < 10000$ and slows down when $V > 10000$. This is because when the system sets a higher V to achieve more profit, *VariedLen* will

transmit more job requests to the system. However, to guarantee queue stability, *VariedLen* has to schedule more containers for job processing under suboptimal situation, thus making the growth slower when $V > 10000$. (2) The time average profit is lower than 0 (i.e., with no profit) when $V = 0$ for *VariedLen*. This is because the drift-minus-profit expression (Eq. (20)) reduces to $\Delta(\Theta(t))$. According to the fronthaul scheduling policy Eq. (27), the system will decline all the requests to minimize the system congestion when $V = 0$.

The proposed *VariedLen* algorithm increases the profit of MSP. As shown in Fig. 5, *VariedLen* outperforms other scheduling methods on profit. However, with respect to the congestion, *VariedLen* will incur higher congestion compared with other four mixture of methods. These method mixtures are *TJF*, *BJF*, *TRF* and *BRF*. Through analyzing, we find the key difference is the mobile server scheduling method. Our *VariedLen* uses a *greedy* scheduling method to balance the profit and congestion instead of the *FCFS* method. The later will execute job requests once they arrive on the server. In this way, the congestion incurred by *FCFS* will be lower than that incurred by the *greedy* method.

We then verify the mobile system stability here. Fig. 6 shows the time average queue congestion [18] captured by Eq. (18) under different V . As shown in the figure, we find: (1) When V increases, the time average queue congestion also increases for *VariedLen*. This phenomenon, together with Fig. 5, reflects the tradeoff between profit maximization and system stability shown in Sec. 3.1.1 under *VariedLen*. (2) Similar to *VariedLen*, the congestions incurred by *TJF*, *TRG* and *TRF* also increase when V increase. while for the other four mixtures of methods (*BJG*, *BJF*, *BRG* and *BRF*), the congestion varies very little when the parameter V changes. The reason is that when the fronthaul scheduling method is *best-effort*, the fronthaul links will transfer job requests to the BBU pool without considering the tradeoff between profit and congestion. Besides, we plot the profit increment of *VariedLen* over other mixture of scheduling policies when $V = 9000$ in Fig. 7.

At last, we plot the decline requests proportion under different V for *VariedLen* compared to other mixture of scheduling algorithms in Fig. 8. As expected, the declined requests decrease when V grows when using the threshold-based policy (*VariedLen*, *TJF*, *TRG*, *TRF*). However, the system still declines a few requests even with an excessive high V . This phenomenon conforms to the fronthaul scheduling policy designed in Sec. 3.2.1. That is, the transmitted requests through the fronthaul links cannot exceed the capacity constraint of each fronthaul links. This can be verified by the declined requests of other mixture of scheduling with *Best-effort* policy (*BJG*, *BJF*, *BRG* and *BRF*) which has the same constraint.

4.4 The Effectiveness of Scheduling policies

As our *VariedLen* algorithm includes three optimal scheduling policies, we evaluate the effectiveness of these three policies here. The first scheduling policy is fronthaul link scheduling, we plot the number of active fronthaul links for *VariedLen* when $V = 10000$, 30000 and 50000 over time slots in Fig. 9. Also, we plot the CDF of active fronthaul links in Fig. 10. From both figures we can see, the fronthaul links are dynamically scheduled in our *VariedLen* and more fronthaul links have been switched to *active* state when V increases. When V is excessively high, *VariedLen* schedules all fronthaul links to *active* state due to the fronthaul scheduling

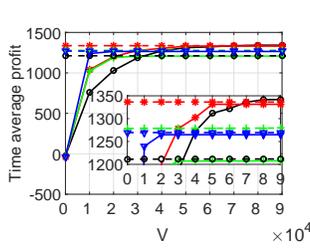


Fig. 5: Time average profit of *VariedLen*, compared to other mixture of scheduling policies. The legends of this figure are the same as Fig. 6.

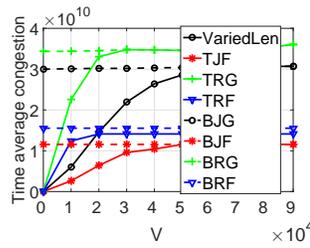


Fig. 6: Time average congestion of *VariedLen*, compared to other mixture of scheduling policies.

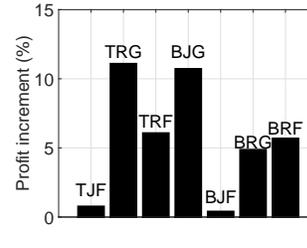


Fig. 7: The profit gain of *VariedLen* over other mixture scheduling policies when $V = 9000$.

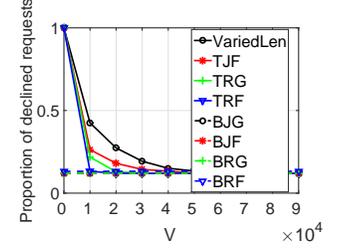


Fig. 8: The proportion of declined requests of *VariedLen*, compared to other mixture of scheduling policies.

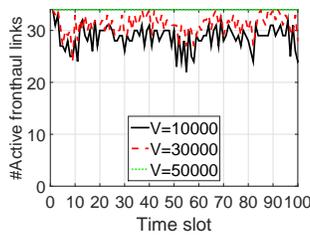


Fig. 9: The number of active fronthaul links under different time slots when $V = 10000$, $V = 30000$ and $V = 50000$ for *VariedLen*.

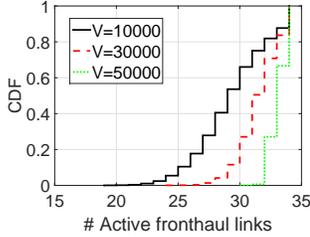


Fig. 10: The CDF of active fronthaul links over time slots when $V = 10000$, $V = 30000$ and $V = 50000$ for *VariedLen*.

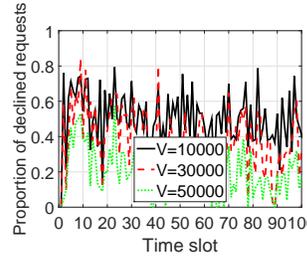


Fig. 11: The proportion of declined requests under different time slots when $V = 10000$, $V = 30000$ and $V = 50000$ for *VariedLen*.

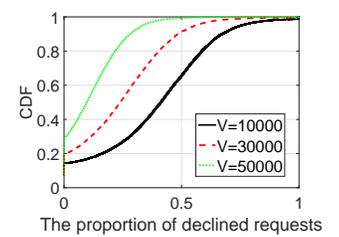


Fig. 12: The CDF of proportion of declined requests over time slots when $V = 10000$, $V = 30000$ and $V = 50000$ for *VariedLen*.

policy in Eq. (27). That is, $X_i(t) < \frac{Vw_i\alpha_i - V\beta}{w_i^2}$ holds for each fronthaul link i .

The fronthaul scheduling will decline some requests from mobile users based on Eq. (27). We plot the proportion of declined requests under different time slots in Fig. 11 and the CDF of proportion of declined requests over time slots in Fig. 12. From the figures we can see, the mobile system declines fewer requests when the V increase. But it still declines requests with a high parameter V due to the capacity limitation of fronthaul links described in Sec. 2.2.

Then we evaluate the server scheduling in edge cloud, we plot the number of *running* containers for *VariedLen* when $V = 10000$, 30000 and 50000 over time slots in Fig. 13. Also, we plot the CDF of *running* containers in Fig. 14. From both figures we can see, more containers are scheduled to *running* state when the time slots increase. That is because the backlog of queue maintained by each container in edge cloud increase and more containers have to be scheduled to *running* state based on the solution described in Sec. 3.2.3.

4.5 Extended vs. Standard Lyapunov Technique

As mentioned in Sec. 3, *VariedLen* has extended the standard Lyapunov technique [18] to deal with job requests with varied lengths. While in our preliminary work [32], we only leverage the standard Lyapunov technique to design an algorithm, i.e., *RICH*, to handle job requests with fixed length. To show the novelty of this paper, we compare the *VariedLen* with the *RICH* by using the same mobile LiveLab trace [20] in this part. Note that the *RICH* simply assumes that each job request can be finished in one time slot and only incurs $\alpha_i \times 1$ revenue for the MSP. But the *VariedLen* has considered the length of each job request. In this way, each job request will incur $\alpha_i \times w_i$ for the MSP. Therefore, we multiply the profit gained by *RICH* with the mean length of all job requests and

plot the result in Fig. 15. From the figure we can see, *VariedLen* achieves about $2 \times$ higher profit than that for *RICH*.

By extending the standard Lyapunov technique, we introduce another parameter, i.e., the time interval length T , when designing the *VariedLen* algorithm in Sec. 3.2. We need to evaluate the sensitivity of the time interval length T on time average profit for *VariedLen*. We plot the time average profit under different control parameters V and time interval T evaluated by *VariedLen* in Fig. 16 and Fig. 17, respectively. From Fig. 16 we can see, the profit with a longer time interval is a little bit higher than that with a shorter time interval. But when the length of a time interval T grows, the gap diminish to zero. While in Fig. 17, for a given value of V , the profit grows very slowly with the growth of time interval T . These two figures suggest that the length of time interval T has little impact on profit, which means that *VariedLen* is not sensitive to the length of time interval T .

4.6 The Sensitivity of Parameters

In this subsection, we show the sensitivity of parameter $\alpha_i, \forall i \in \mathcal{U}$, β , γ and T in time average profit maximization Problem (P) for *VariedLen* algorithm. In previous evaluations, we set all the parameters α_i the same (i.e., all equal to 1), which means that we treat all user the same. But in reality, we can flexibly assign different users by choosing appropriate values of α_i , as discussed in Sec. 2.4.3.

We choose to change the values of half of α_i and compare the profit under different situations to show the effectiveness of our fronthaul scheduling. We have set four types of parameters, Type 1 is the same as previous evaluations, and Type 2 denotes as $\alpha_i = (1, 2, \dots, 1, 2)$ while Type 3 for $\alpha_i = (1, 4, \dots, 1, 4)$ and Type 4 for $\alpha_i = (1, 8, \dots, 1, 8)$. The effectiveness of our fronthaul scheduling can be illustrated by comparing MSPs profits under different types of α_i . For example, we expect to achieve the

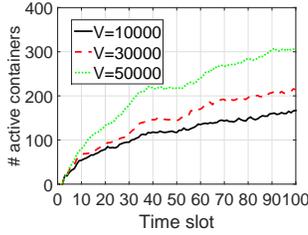


Fig. 13: The number of *running* containers under different time slots when $V=10000$, $V=30000$ and $V=50000$ for *VariedLen*.

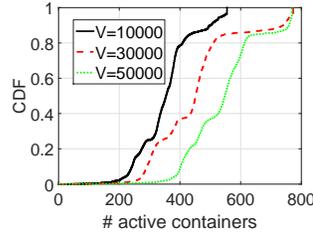


Fig. 14: The CDF of *running* containers over time slots when $V=10000$, $V=30000$ and $V=50000$ for *VariedLen*.

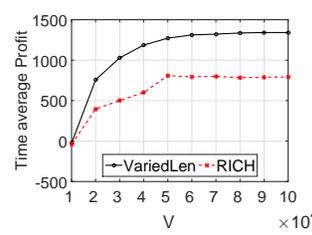


Fig. 15: Time average profit of *VariedLen*, compared to the *RICH* which uses standard Lyapunov technique.

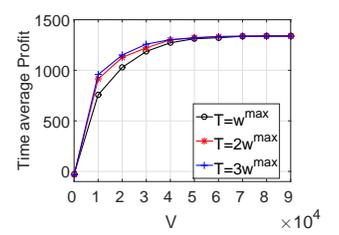


Fig. 16: Time average profit under different control parameter V for *VariedLen*.

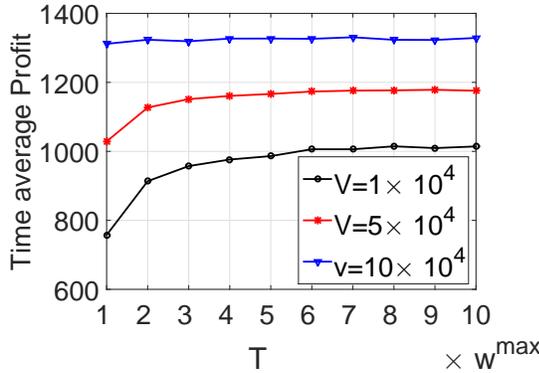


Fig. 17: Time average profit under different time interval T for *VariedLen*.

profit of Type 2 as $(1+2)/2 = 1.5\times$ than that of Type 1. However, the experiment result shown in Fig. 18 is about $2\times$. However, the experiment result shown in Fig. 18 is about $2\times$. Similar results appear for Type 3 and Type 4. The reason is that the system will schedule fronthaul links to transmit more requests with higher α_i , e.g., higher priorities or prices, to improve the profit as expected.

Then, we evaluate the sensitivity of parameter β , and plot the time average profit for various values of β in Fig. 19. From the figure we can see, the profit decreases with the growth of β when the control parameter V is given. This reflects that more power consumption needed for the fronthaul, less profit achieved by the MSP. Meanwhile, in order to achieve the maximum level of profit, MSP has to set a larger V when β grows. This reflects the importance of fronthaul in C-RAN system.

We plot the time average profit for various values of γ in Fig. 20. It can be seen that the time average profit decreases when γ grows for a given V . Note that the parameter γ equals to $Price \times PUE_s$ in Sec. 2.4.2. Therefore, Fig. 20 reveals that the time average profit decreases when the electricity market price increases. Meanwhile, the system can improve the PUE of servers to achieve higher profit.

At last, we evaluate the sensitivity of parameter η , and plot the time average profit for different η in Fig. 21. As can be seen from the figure, the profit increase with the growth of η when the control parameter V is given. Recall that $1 - \eta$ denotes the idle server's power consumption. Therefore, Fig. 21 indicates that one can reduce the power consumption by increasing the parameter η , i.e., to design more power efficient servers which consume less power in its idle state. According to the above evaluation results, MSPs can set proper values for each parameter in *VariedLen* algorithm.

5 RELATED WORK

In order to cope with the growth of mobile traffic, mobile edge computing (or Fog computing) has been proposed and received much interest in the literature [10], [13], [14]. For example, in [10], a game theory had been used to efficiently schedule the computation resource in MEC for multi-user. Also, many surveys on MEC have emerged [45], [46]. In [45], a research outlook with an integration of mobile computing and wireless communications in MEC had been discussed.

At the same time, C-RAN had been presented as a new promising network and received much interest in both industry and academia [3], [4], [47]. However, there are fewer studies of integration between MEC and C-RAN. Cai et al [27] had studied the topology configuration and rate allocation in C-RAN with the objective of optimizing the end-to-end TCP throughput performance of mobile computing. A cross-layer resource allocation model for C-RAN to minimize the overall system power consumption in both the BBUs and RRHs had been investigated [29]. But none of them has considered the power-performance tradeoff in the mobile system consisting of C-RAN and MEC. In this paper, we design the *VariedLen* algorithm to optimize the power-performance tradeoff by using the Lyapunov optimization [18]. We dynamically schedule resources including fronthaul links in C-RAN, the Dispatcher in BBU pool and mobile servers in edge cloud to achieve a time average profit maximization. The key difference between this work and other studies is that we conduct research at different levels. For example, Cai et al [27] focused on the TCP network from the physical layer and optimized the TCP throughput performance. Tang et al [29] focused on a cross-layer resource allocation, including the incoming traffic rate from the application layer and wireless channel state information from the physical layer. While in our work, we mainly focus on the application level with respect the throughput of the whole system of MSP. Hence, our work is orthogonal to these studies.

It is worth noting that MEC has a strong relationship with another mobile computing paradigm (i.e., MCC). MCC has also gained a lot of attention in recent years [48], [49]. For example, Kumar et al [7] had investigated the feasibility for saving energy and extending battery lifetimes by offloading tasks to the cloud. Zhang et al [49] considered the energy optimization under stochastic wireless channel with theoretical framework while Chen [48] investigated efficient offloading by utilizing game theoretical approach in MCC.

There also exist many online solutions [19], [22], [30], [34], [36] for power management and dynamic resource allocation in datacenters. For example, in SaaS cloud, Zhou et al [22] investigated the power management in the datacenter by using

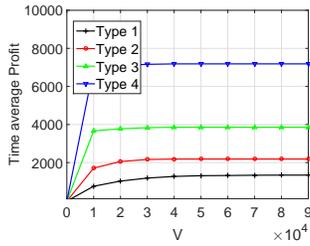


Fig. 18: Time average profit under different control parameter V with different values of α for *VariedLen*.

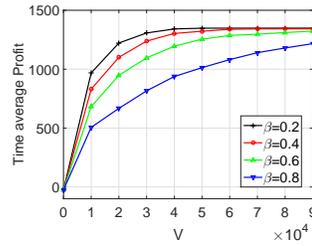


Fig. 19: Time average profit under different control parameter V with different values of β for *VariedLen*.

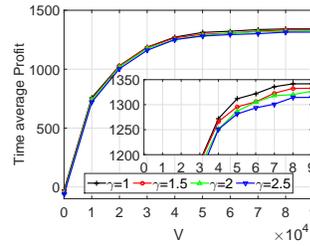


Fig. 20: Time average profit under different control parameter V with different values of γ for *VariedLen*.

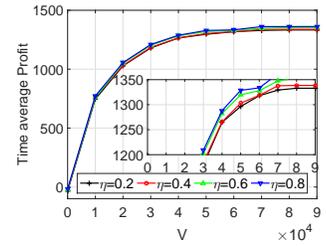


Fig. 21: Time average profit under different control parameter V with different values of η for *VariedLen*.

admission control and request routing approaches while Xiang et al [36] studied the problem of greening the SaaS clouds by VM scheduling and routing in both intra- and inter-datacenter in a geo-distributed scenario. In [19], online algorithm had been designed to dynamically price the VM resources, schedule jobs and provision servers across datacenters in a geo-distributed cloud.

Different from these work mentioned above, we particularly adjust Lyapunov optimization technique for dynamic resource scheduling in the context of the mobile system with C-RAN and MEC. After transforming the original optimization problem to new problems, we have designed scheduling policies for fronthaul links, the Dispatcher and edge servers in the whole mobile system. The scheduling policy of fronthaul is a l_0 -norm problem which is hard to solve, so we use l_1 -norm to relax it. Meanwhile, for the power usage, we take the fronthaul power introduced in C-RAN into consideration along with the servers in edge cloud. Meanwhile, we have extended the standard Lyapunov technique to handle the situation that users' job can exceed the length of the online decision making interval which cannot be handled by the standard Lyapunov technique. The proposed algorithm, *VariedLen*, can make decisions in consecutive time slots and still ensure its performance that is close to the optimum.

While in conventional cloud datacenter, there exist many studies [31], [39], [50], [51] for power management and dynamic resource allocation. In [31], a dynamic speed scaling method had been proposed to reduce the energy consumption. While in a cloud environment, Zhang et al [50] proposed a dynamic and heterogeneity-aware capacity provisioning for resource management. Maguluri et al [39] considered the resource allocation problems with a stochastic model in the cluster computing. Liu et al [51] integrated renewable supply, dynamic pricing and cooling supply to reduce the electricity cost and environmental impact.

6 CONCLUSION

In this paper, we propose an unifying optimization framework for maximizing the profit of MSP. The framework can jointly schedule computation resources in MEC and network resources in C-RAN, and handle dynamic and unpredictable requests of mobile users due to their mobilities. Specially, we allow job requests from different mobile users have variable lengths which can not be handled by the standard Lyapunov technique. Hence, we extend the standard Lyapunov technique to design the *VariedLen* algorithm which consists a threshold-based scheduling policy for the fronthaul links, a load balancing policy for request dispatching in the BBU Dispatcher and a greedy scheduling policy to optimally schedule the containers. Our algorithm can achieve time average profit which is close to the optimum for MSP, while the system stability is still strong.

In the future, we would like to extend our research in two directions. The first direction is to consider the interference of fronthaul transmission among all users. The other one is to design a pricing strategy of mobile users to process their requests in MEC. Now we only use tunable parameters for each mobile users in this paper. While in the future, we can design a more sophisticated pricing strategy for the mobile system.

ACKNOWLEDGE

This work was supported by National Key Research and Development Program under grant 2016YFB1000501, National Science Foundation of China under grant No. 61472151, the Fundamental Research Funds for the Central Universities under grants 2015TS067 and 2016YXZD016, Chinese Universities Scientific Fund under grant 2013TS094, International Science & Technology Cooperation Program of China under grant No. 2015DFE12860 and also supported partially by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research Program, under Contract DE-AC02-06CH11357, EU FP7 Project MONICA under grant PIRSES-GA-2011-295222.

REFERENCES

- [1] Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2012-2017, February 2014.
- [2] China mobile research institute, C-RAN white paper: The road towards green Ran, <http://labs.chinamobile.com/cran>, June 2014.
- [3] J. Wu, "Green wireless communications: from concept to reality," *IEEE Wireless Communications*, vol. 19, no. 4, pp. 4–5, August 2012.
- [4] X. Rao and V. Lau, "Distributed fronthaul compression and joint signal recovery in Cloud-RAN," *IEEE Transactions on Signal Processing*, vol. 63, no. 4, pp. 1056–1065, February 2015.
- [5] 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. of MobiSys*, 2010, pp. 49–62.
- [6] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *Proc. of INFOCOM*, March 2012, pp. 945–953.
- [7] K. Kumar and Y. Lu, "Cloud computing for mobile users: Can offloading computation save energy?" *Computer*, vol. 43, no. 4, pp. 51–56, April 2010.
- [8] Google compute engine, <https://cloud.google.com/pricing/compute-engine>, 2017.
- [9] Amazon elastic compute cloud (ec2), <http://aws.amazon.com/ec2/>, 2017.
- [10] 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.
- [11] Mobile-edge computing-introductory technical white paper, *European Telecommunications Standards Institute*, 2017.
- [12] A. Checko, H. Christiansen, Y. Yan, L. Scolari, G. Kardaras, M. Berger, and L. Dittmann, "Cloud RAN for mobile networks: A technology overview," *IEEE Communications Surveys Tutorials*, vol. 17, no. 1, pp. 405–426, Firstquarter 2015.

- [13] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, "Mobile-edge computing: Partial computation offloading using dynamic voltage scaling," *IEEE Transactions on Communications*, vol. 64, no. 10, pp. 4268–4282, Oct 2016.
- [14] L. Gu, D. Zeng, S. Guo, A. Barnawi, and Y. Xiang, "Cost efficient resource management in fog computing supported medical cyber-physical system," *IEEE Transactions on Emerging Topics in Computing*, vol. 5, no. 1, pp. 108–119, Jan 2017.
- [15] K. Wang, K. Yang, H. Chen, and L. Zhang, "Computation diversity in emerging networking paradigms," *IEEE Wireless Communications*, vol. 24, no. 1, pp. 88–94, 2017.
- [16] W. Li, Y. Zhao, S. Lu, and D. Chen, "Mechanisms and challenges on mobility-augmented service provisioning for mobile cloud computing," *IEEE Communications Magazine*, vol. 53, no. 3, pp. 89–97, March 2015.
- [17] Docker, <http://www.docker.com/>, 2017.
- [18] M. J. Neely, *Stochastic network optimization with application to communication and queueing system*, Morgan & Claypool, 2010.
- [19] J. Zhao, H. Li, C. Wu, Z. Li, Z. Zhang, and F. C. M. Lau, "Dynamic pricing and profit maximization for the cloud with geo-distributed data centers," in *Proc. of INFOCOM*, April 2014, pp. 118–126.
- [20] C. Shepard, A. Rahmati, C. Tossell, L. Zhong, and P. Kortum, "Livelab: Measuring wireless networks and smartphone users in the field," *SIGMETRICS Perform. Eval. Rev.*, vol. 38, no. 3, pp. 15–20, Jan. 2011.
- [21] S. Ren and M. van der Schaar, "Dynamic scheduling and pricing in wireless cloud computing," *IEEE Transactions on Mobile Computing*, vol. 13, no. 10, pp. 2283–2292, October 2014.
- [22] Z. Zhou, F. Liu, H. Jin, B. Li, B. Li, and H. Jiang, "On arbitrating the power-performance tradeoff in SaaS clouds," in *Proc. of INFOCOM*, April 2013, pp. 872–880.
- [23] L. Yang, J. Cao, S. Tang, T. Li, and A. Chan, "A framework for partitioning and execution of data stream applications in mobile cloud computing," in *Proc. of CLOUD*, 2012, pp. 794–802.
- [24] K. Wang, K. Yang, and C. Magurawalage, "Joint energy minimization and resource allocation in c-ran with mobile cloud," *IEEE Transactions on Cloud Computing*, vol. PP, no. 99, pp. 1–1, 2016.
- [25] B. Dai and W. Yu, "Sparse beamforming and user-centric clustering for downlink cloud radio access network," *IEEE Access*, vol. 2, pp. 1326–1339, 2014.
- [26] S. Wu, C. Niu, J. Rao, H. Jin, and X. Dai, "Container-based cloud platform for mobile computation offloading," in *Proc. of IPDPS*, May 2017, pp. 1–10.
- [27] Y. Cai, F. Yu, and S. Bu, "Dynamic operations of cloud radio access networks (C-RAN) for mobile cloud computing systems," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 3, pp. 1536–1548, March 2016.
- [28] T. Chen and R. Bahsoon, "Self-adaptive trade-off decision making for autoscaling cloud-based services," *IEEE Transactions on Services Computing*, vol. 10, no. 4, pp. 618–632, July 2017.
- [29] J. Tang, W. P. Tay, and T. Quek, "Cross-layer resource allocation with elastic service scaling in cloud radio access network," *IEEE Transactions on Wireless Communications*, vol. 14, no. 9, pp. 5068–5081, September 2015.
- [30] Y. Yao, L. Huang, A. Sharma, L. Golubchik, and M. Neely, "Power cost reduction in distributed data centers: A two-time-scale approach for delay tolerant workloads," *IEEE Transactions on Parallel Distribution System*, vol. 25, no. 1, pp. 200–211, 2014.
- [31] A. Wierman, L. Andrew, and A. Tang, "Power-aware speed scaling in processor sharing systems," in *Proc. of INFOCOM*, April 2009, pp. 2007–2015.
- [32] X. Wang, K. Wang, S. Wu, S. Di, K. Yang, and H. Jin, "Dynamic resource scheduling in cloud radio access network with mobile cloud computing," in *Proc. of IWQoS*, June 2016, pp. 1–6.
- [33] H. Jin, X. Wang, S. Wu, S. Di, and X. Shi, "Towards optimized fine-grained pricing of IaaS cloud platform," *IEEE Transactions on Cloud Computing*, vol. 3, no. 4, pp. 436–448, October 2015.
- [34] R. Uргаonkar, U. Kozat, K. Igarashi, and M. Neely, "Dynamic resource allocation and power management in virtualized data centers," in *Proc. of NOMS*, April 2010, pp. 479–486.
- [35] K. Wang, K. Yang, X. Wang, and C. Magurawalage, "Cost-effective resource allocation in c-ran with mobile cloud," in *Proc. of ICC*, 2016, pp. 1–6.
- [36] X. Xiang, C. Lin, F. Chen, and X. Chen, "Greening geo-distributed data centers by joint optimization of request routing and virtual machine scheduling," in *Proc. of UCC*, 2014, pp. 1–10.
- [37] J. Zhao, T. Quek, and Z. Lei, "Coordinated multipoint transmission with limited backhaul data transfer," *IEEE Transactions on Wireless Communications*, vol. 12, no. 6, pp. 2762–2775, June 2013.
- [38] J. Tang, W. Tay, and T. Quek, "Cross-layer resource allocation in cloud radio access network," in *Proc. of GlobalSIP*, Dec 2014, pp. 158–162.
- [39] S. Maguluri, R. Srikant, and L. Ying, "Stochastic models of load balancing and scheduling in cloud computing clusters," in *Proc. of INFOCOM*, March 2012, pp. 702–710.
- [40] D. Hayes, D. Ros, A. Petlund, and I. Ahmed, "A framework for less than best effort congestion control with soft deadlines," in *Proc. of IFIP Networking*, 2017, pp. 1–9.
- [41] A. Sirohi, A. Pratap, and M. Aggarwal, "Improvised round robin (cpu) scheduling algorithm," *International Journal of Computer Applications*, vol. 99, no. 18, pp. 40–43, 2014.
- [42] R. Frijns, S. Adyanthaya, S. Stuijk, J. Voeten, M. Geilen, R. Schifferers, and H. Corporaal, "Timing analysis of first-come first-served scheduled interval-timed directed acyclic graphs," in *Proc. of DATE*, 2014, pp. 288:1–288:6.
- [43] Y. Nan, W. Li, W. Bao, F. Delicato, P. Pires, Y. Dou, and A. Zomaya, "Adaptive energy-aware computation offloading for cloud of things systems," *IEEE Access*, vol. 5, pp. 23 947–23 957, 2017.
- [44] Y. Nan, W. Li, W. Bao, F. Delicato, P. Pires, and A. Zomaya, "A dynamic tradeoff data processing framework for delay-sensitive applications in cloud of things systems," *Journal of Parallel and Distributed Computing*, vol. 112, pp. 53–66, Oct 2017.
- [45] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "Mobile edge computing: Survey and research outlook," *arXiv:1701.01090*, pp. 1–30, 2017.
- [46] Y. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing - A key technology towards 5G," *ETSI White Paper*, vol. 11, 2015.
- [47] D. Xu, P. Ren, Q. Du, L. Sun, and Y. Wang, "Towards win-win: weighted-voronoi-diagram based channel quantization for security enhancement in downlink cloud-ran with limited csi feedback," *SCIENCE CHINA Information Sciences*, vol. 60, no. 4, 2017.
- [48] X. Chen, "Decentralized computation offloading game for mobile cloud computing," *Transactions on Parallel and Distributed Systems*, vol. 26, no. 4, pp. 974–983, April 2015.
- [49] W. Zhang, Y. Wen, K. Guan, D. Kilper, H. Luo, and D. Wu, "Energy-optimal mobile cloud computing under stochastic wireless channel," *IEEE Transactions on Wireless Communications*, vol. 12, no. 9, pp. 4569–4581, September 2013.
- [50] Q. Zhang, M. Zhani, R. Boutaba, and J. Hellerstein, "Harmony: Dynamic heterogeneity-aware resource provisioning in the cloud," in *Proc. of ICDCS*, July 2013, pp. 510–519.
- [51] Z. Liu, Y. Chen, C. Bash, A. Wierman, D. Gmach, Z. Wang, M. Marwah, and C. Hysler, "Renewable and cooling aware workload management for sustainable data centers," in *Proc. of SIGMETRICS*, 2012, pp. 175–186.