Stateful Serverless Application Placement in MEC With Function and State Dependencies

Zichuan Xu, *Member, IEEE*, Lizhen Zhou, Weifa Liang, *Senior Member, IEEE*, Qiufen Xia, *Member, IEEE*, Wenzheng Xu, *Member, IEEE*, Wenhao Ren, Haozhe Ren, and Pan Zhou, *Senior Member, IEEE*

***Abstract*—Serverless computing is emerging as an enabling technology for elastic and low-cost AI applications in the edge of core networks. It allows AI developers to decompose a complex training and time-sensitive inference task into multiple functions with dependency, and upload the task to a Multi-access Edge Computing platform (MEC) for execution. Serverless computing adopts a popular design principle: the disaggregation of storage and computation, making the functions 'stateless'. However, most AI applications are 'stateful' and rely on an external storage service to manage their states (ephemeral data). This will incur a prohibitively long delay for delay-sensitive AI applications if external services storing the states are far from the serverless functions. Motivated by this critical issue, in this paper we investigate a fundamental problem in serverless computing – the stateful serverless application placement problem, for which, we first propose an efficient heuristic algorithm, and then devise an approximation algorithm with a provable approximation ratio for one of its special cases. We also consider the online version of the problem, and develop an online learning-driven algorithm with a bounded regret. The crux of the online algorithm is the adoption of the multi-armed bandits technique for dynamic admissions of inference requests, under the uncertainty of both data volumes of requests and network delays. We finally evaluate the performance of the proposed algorithms through experimental simulations. Simulation results show that the proposed algorithms outperform their counterparts, reducing at least 32% in the total cost and 27% of the average delay.**

***Index Terms*—Cloud computing, function placement, machine learning method, multi-access edge computing, online learning, serverless computing, the multi-armed bandit method.**

Manuscript received 26 April 2022; revised 2 September 2022; accepted 9 March 2023. Date of publication 29 March 2023; date of current version 9 August 2023. The work of Zichuan Xu and Qiufen Xia was supported in part by the National Natural Science Foundation of China (NSFC) under Grants 62172068 and 62172071, and the "Xinghai scholar" program. The work of Weifa Liang was supported in part by a grant from City University of Hong Kong under Grant 9380137/CS. The work of Wenzheng Xu was supported in part by NSFC under Grant 62272328. The work of Pan Zhou was supported in part by NSFC under Grant 61972448. Recommended for acceptance by M. T. Kandemir. *(Corresponding author: Wenzheng Xu.)*

Zichuan Xu, Lizhen Zhou, Wenhao Ren, and Haozhe Ren are with the School of Software, Dalian, Liaoning 116024, China (e-mail: z.xu@dlut.edu.cn; zhou_lizhen@mail.dlut.edu.cn; hangvane@mail.dlut.edu.cn; renhaozhe@mail.dlut.edu.cn).

Weifa Liang is with the Department of Computer Science, City University of Hong Kong, Hong Kong (e-mail: weifa.liang@cityu.edu.hk).

Qiufen Xia is with the International School of Information Science and Engineering, Key Laboratory for Ubiquitous Network and Service Software of Liaoning Province, Dalian University of Technology, Dalian, Liaoning 116024, China (e-mail: qiufenxia@dlut.edu.cn).

Wenzheng Xu is with the Department of Computer Science, Sichuan University, Chengdu, Sichuan 610000, China (e-mail: wenzheng.xu@scu.edu.cn).

Pan Zhou is with the School of Cyber Science and Engineering, Huazhong University of Science and Technology, Wuhan, Hubei 430074, China (e-mail: panzhou@hust.edu.cn).

Digital Object Identifier 10.1109/TC.2023.3262947

I. Introduction

CLOUD service developers now spend a huge amount of time for the management of virtual machines (VMs). Meanwhile, cloud service providers are seeking for an agile and low-cost computing paradigm to release themselves from time-consuming VM management and accelerate innovation in developing AI applications at network edges, such as smart speakers, self-driven cars, and intelligent surveillance cameras. Serverless computing is emerging as the next-generation computing paradigm that enables developers to write serverless code as *functions* for AI applications, without having to provision and maintain VMs [5], [9], [28], [32]. In 2021, 36% of data analytic applications and 27% of streaming media applications in many edge/cloud enterprises have adopted serverless computing, 83% developers are likely to adopt serverless computing to develop their applications in the next two years, and a quarter of organizations that use Amazon CloudFront are using Lambda@Edge to move applications to the edge [34]. The key technology of serverless computing is function-as-a-service (FaaS), where applications are implemented as a set of dependent functions running in containers. A serverless-enabled multi-access edge computing (MEC) network receives the submitted functions from developers and places them into cloudlets within the proximity of users.

The key to enabling serverless computing in the network edge for AI applications is to place stateful serverless functions. Specifically, each AI application generates various states that need to be shared among its functions. For instance, stateful functions keep track of data processing sessions or transactions and react differently to the same input with different sessions. Such states usually are shared with the other functions. However, existing serverless computing platforms are designed to support stateless functions only, by adopting a method of 'compute and storage separation' [4], [23]. Thus, direct deployment of AI applications into serverless computing platforms will lead to the separate storage of state and placement of functions, i.e., the state of an AI application is directly stored to a separate storage system for the usage of other functions of the application [6], [33]. Consequently, other functions have to fetch the states from their stored cloudlets that may be far away from them, leading to a long delay [33]. To avoid such a performance degradation, in this paper we investigate the joint placement of serverless functions and their states.

Enabling stateful serverless application placement in MEC poses several challenges. First, there are complex dependencies

between functions and their states in AI applications, where a function requires both the input data processed by the other functions and their states to guarantee the correct processing of its data. Jointly considering the dependency of both functions and their states in their placements is challenging. Second, AI applications usually have stringent delay requirements on the processing of their data streams. The delay of implementing an application depends on interplays between data processing and state fetching of serverless functions, while meeting the delay requirements with function and state dependencies is extremely challenging. Finally, serverless computing allows developers to directly submit their code (functions) for execution. As such, the data volume of each application request usually is unknown in advance, so are the transmission and processing delays experienced, and such an uncertainty needs to be learnt so that the delay requirement of each request can be met. However, placing functions under such an uncertainty significantly impacts the system performance, thereby generating 'regrets' in decision makings. Therefore, how to make online learning-driven decisions on the placement of functions and their states with a bounded regret is challenging.

To the best of our knowledge, we are the first to investigate the problem of stateful serverless application placements in MEC, by considering not only the dependency of functions and states of serverless applications but also the uncertainty of both data volumes and network delays. Although there are several studies on service placements/migrations and function placements for serverless applications, most of them either ignored function and state dependencies [11], [13], [26], or assumed that the data volume of requests of each application is given in advance [15], [23], [26].

The main contributions of this paper are as follows.

- We formulate a novel optimization problem of placing stateful serverless applications in MEC networks.
- We devise an efficient heuristic for the problem of stateful serverless application placement. We also develop an approximation algorithm with a provable approximation ratio for a special case of the problem.
- We propose an online learning-driven algorithm with a bounded regret for the dynamic stateful serverless application placement problem under the uncertainty assumption of both data volumes and network delays.
- We evaluate the performance of the proposed algorithms by experimental simulations. Simulation results show that the performance of the algorithms outperform their counterparts, by reducing at least 32% in the total cost and 27% in the average delay.

The rest of this paper is organized as follows. Section II summarizes state-of-the-arts on the topic. Section III introduces the system model and formulates the optimization problems. Section IV proposes a heuristic algorithm for the problem and an approximation algorithm for a special case of the problem, respectively. Section V devises an online learning-driven algorithm for the dynamic stateful serverless application placement problem. Section VI provides experimental results on the performance of the proposed algorithms, and the paper is concluded in Section VII.

II. Related Work

Although service placement, VNF placement, migration, and caching in MEC have been intensively investigated in the past several years [11], [17], [18], [21], [25], [31], [41], [42], [45], none of the studies considered the placement of serverless applications with dependency between the components of applications: functions and their state dependencies. For example, Gao et al. [17] considered the problem of joint network selection and service placement in MEC with the aim to improve the switching and communication delays of mobile users. Liang et al. [21] studied the problem of service entity caching to minimize the cost of network service providers. Ma et al. [25] dealt with the problem of mobility-aware service placement in MEC networks. Zhang et al. [45] studied the service placement problem to minimize the placement cost while meeting performance requirements.

There are several recent studies on the serverless function placement and resource provisioning [3], [7], [8], [10], [14], [15], [22], [27], [29], [30], [35], [36], [39], [44], none of these studies considered state management of serverless applications. For instance, Akhtar et al. [3] adopted statistical learning techniques to predict the cost and execution time of each serverless function. Chaudhry et al. [10] presented an approach to deploy virtualized network functions to serverless platforms. Elgamal et al. [15] studied the problem of function fusion and placement in MEC. Mahmoudi et al. [27] proposed an adaptive function placement algorithm, using machine learning to jointly optimize the system performance and the service cost. Palade et al. [29] investigated a swarm-based approach for placing functions with a federated view of available edge resources. Pelle et al. [30] presented a framework for automated deployments and reconfigurations of latency-sensitive serverless applications. Xu et al. [39] provided a resource provisioning framework for AI applications to minimize the cost of executing serverless functions. Zhang et al. [44] aimed to speed up the invocation process of serverless functions, by harvesting the spare cycles on servers to warm up functions.

Despite that a few studies [6], [43] investigated stateful serverless applications that require fine-grained support for state management and synchronization, none of them considered the placement of serverless applications with function and state dependencies. Bhasi et al. [8] investigated the problem of resource allocation for serverless functions while ensuring service level objectives, through determining the number of containers needed for the functions along each path in the DAG of an application. Daw et al. [14] aimed to reduce the overhead due to cold-starts of serverless functions, by detecting the most-likely-path in a function workflow and pre-provisioned the execution sandboxes. It must be mentioned that although the studies in [8], [14] considered function dependency, they did not incorporate the state dependency into the function dependency for consideration. In contrast, we jointly considered the dependency of functions and states. Besides, they mainly focused on the prediction function execution times to optimize the configuration and cold-start of functions. However, our focus is to optimize the placement of serverless functions. Although Mahgoub et al. [26]

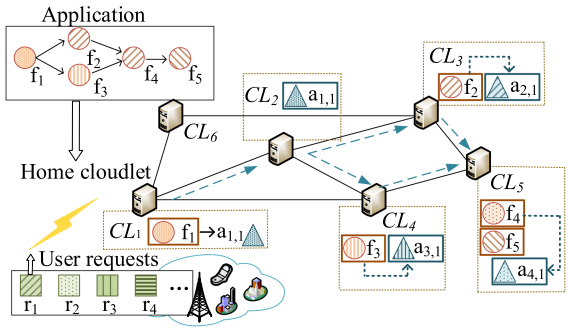


Fig. 1. The MEC network G, state $a_{1,1}$ is generated by function f_1 during the processing of request r_1. $a_{1,1}$ is stored in CL_2 for the usage of its demanded functions. Also, states $a_{2,1}$, $a_{3,1}$, and $a_{4,1}$ are stored in CL_3, CL_4, CL_5 respectively, for function f_5 to implement r_1.

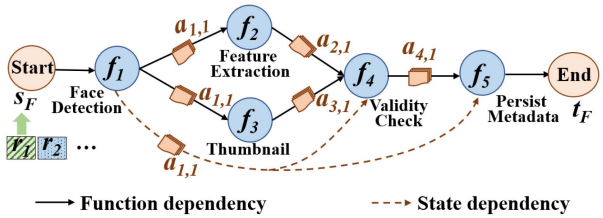


Fig. 2. State and function dependencies of an image processing application. Note that the state generated by each function may not only be needed by its successors, but also by other functions in the application.

considered function and state dependency, they assumed that the state of a function can only be demanded by its successors in the directed acyclic graph (DAG) of a serverless application. However, the state of a function may be demanded by not just its successors. Further, the uncertainty of both data volumes and delays is ignored.

III. Preliminaries

In this section, we first introduce the system model, notions and notations. We then define the optimization problems.

A. System Model

We consider an MEC network $G = (\mathcal{CL}, \mathcal{E})$ consisting of a set $\mathcal{CL}$ of cloudlets deployed within the proximity of users and a set $\mathcal{E}$ of links/paths between cloudlets. Let $CL_i \in \mathcal{CL}$ be a cloudlet. Due to the space, heat dissipation, and load-bearing limitations of edge locations, each cloudlet CL_i contains several servers and thus its computing resource is capacitated. Following several commercial serverless platforms [34], we assume that the memory resource is the major resource demand of serverless applications, as the amount of memory allocated to a serverless function determines how much data can be loaded into the memory at the same time. Note that the amount of computing resource assigned to a function is proportional to the amount of memory allocated to the function [15]. Let $M(CL_i)$ be the memory capacity of cloudlet CL_i. Each link/path $e \in \mathcal{E}$ incurs transmission cost and delay when transmitting data. Let d_e and c_e be the transmission delay and cost for transmitting a unit of data along link e, respectively. Fig. 1 is an illustrative example of the MEC.

B. Stateful Serverless Applications and User Requests

Serverless computing allows developers to directly submit code in terms of functions to cloudlets for executions. A serverless application usually is considered as a series of functions used as event triggers to process data, such as face recognition and speech recognition. When different users use the same type of serverless application, they issue different user requests to process various data. That is, each serverless application may be required by multiple user requests with each having different amount of data volume for processing. Let F be a serverless application. Denote by R the set of requests that need to be implemented by application F, and let r_j be a request in R. where ρ_j represents the data volume generated by request r_j. Once deployed in the MEC, a serverless application can be allocated a certain amount of memory resource to process the data of user requests continuously. We assume that each application F has a *home cloudlet* that not only stores its data but also serves as the *portal* of its user requests. To process its data, each request r_j accesses the MEC via a source node v_j (usually an access point). Its data is then transferred from its source v_j to the home cloudlet of its application F for processing.

C. Function and State Dependencies

Each serverless application F is implemented as a composition of multiple functions, and each function needs to be invoked multiple times. Let f_l be a function in F. Denote by N_l the number of invocations of function f_l to implement a user request. Each application has the dependency of both functions and states. Serverless application F thus is represented by a DAG $G_f = (V_f, E_f)$, where each vertex in G_f represents a function, and each link $\langle f_l, f_{l'} \rangle$ in E_f represents a directed edge from function vertex f_l to function vertex $f_{l'}$ that represents the function dependency, i.e., f_l has to be executed before $f_{l'}$, and the processed data by f_l needs to be passed onto $f_{l'}$.

Serverless application F generates various states when invoking its functions [6], [33], [43]. Further, the state generated by each function f_l in F may be needed by the other functions in F (not necessarily its successors in G_f). For example, Fig. 2 shows an image processing application that is implemented by serverless functions, and the state data generated by 'face detection' includes Exchangeable Image File format (EXIF) data. Such state data with EXIF data needs to be passed to not only its successors 'feature extraction' and 'thumbnail' but also the other functions 'validity check' and 'persist metadata' as well. Let $a_{l,j}$ be the state generated by f_l during the processing of request r_j. As mentioned in [26], the state $a_{l,j}$ is proportional to its original data volume ρ_j. That is, $a_{l,j} = \rho_j \cdot \eta_l$, where η_l is a proportional constant parameter in the range of $(0, 1)$. Fig. 2 gives an example of a serverless application F with the dependency of both functions and states.

D. Delay Model

Each request $r_j \in R$ usually has a delay requirement d_j^{req} to guarantee that its data volume can be processed timely, and the delay of r_j thus has processing and transmission delays.

The processing delay experienced by request r_j depends on both the data volume that needs to be processed, the number of invocations of each function, and the memory resource allocated to process the data. The reason is that with more memory resource allocated to the request, more of its data can be loaded to the memory at the same time, and thus reducing the overhead caused by memory management of operating systems (such as swapping). If a new container has been instantiated for a function, the processing delay may also consist of the cold-start delay, which is dominated primarily by the library download and setup delay and process startup delay [33]. Let $d^o_{l,i}$ be the cold-start delay incurred by creating a container instance for function f_l in CL_i, which is a given constant. We use $x_{j,l,i}$ to indicate whether a newly-instantiated container in cloudlet CL_i is used to implement function f_l of request r_j. On the other hand, we also use $x'_{j,l,i}$ to show whether f_l of r_j is assigned to an existing container of cloudlet CL_i. Denote by $d^{proc}_{l,j}(CL_i)$ the processing delay of request r_j by function f_l in cloudlet CL_i. Considering that different memory configurations (i.e., 128 MB or 256 MB) lead to different processing delays of each function [3], [19], we then have

$$d^{proc}_{l,j}(CL_i) = (x_{j,l,i} + x'_{j,l,i}) \cdot \alpha_i \cdot M_i \cdot N_l \cdot \rho_j + x_{j,l,i} \cdot d^o_{l,i}, \tag{1}$$

where α_i is an influence factor that captures cloudlet CL_i's impact of memory allocation on the processing delay, and M_i is the amount of memory allocated by cloudlet CL_i to process a unit volume of data. Since each function f_l can be assigned to either an existing or a newly-instantiated container in a cloudlet of the MEC network, we then have

$$x_{j,l,i} + x'_{j,l,i} \leq 1. \tag{2}$$

There exists multiple execution paths in DAG G_f from its entry function s_F to its exit function t_F in the DAG G_f. Let $\mathcal{P}'$ be the execution paths from start point s_F to the end point t_F in G_f, and p' be one of such paths. Note that an execution path p' in G_f includes the transmission of both original data and state of each request. Denote by p a path in the MEC network to which the execution path p' in G_f is mapped, and the set of such paths is $\mathcal{P}$. Let $z_{p',p}$ be the indicator variable that shows whether $p' \in \mathcal{P}'$ is mapped to $p \in \mathcal{P}$. The sum of processing and transmission delay of request r_j is the maximum delay of the execution paths in $\mathcal{P}'$ incurred by the execution of the functions and transmission of the data and state in paths of $\mathcal{P}$. Denote by d_j the sum of processing and transmission delay of each request r_j, we have

$$d_j = \max_{p' \in \mathcal{P}'} \sum_{p \in \mathcal{P}} z_{p',p} \left(\sum_{f_l \in p'} \sum_{CL_i \in p} d^{proc}_{l,j}(CL_i) + \sum_{l=1}^{|p'|-1} \sum_{p_s \subseteq p} x_{j,l,\delta(p_s)} \cdot x_{j,l+1,\psi(p_s)} \sum_{e \in p_s} (\rho_j + a_{l,j})\, d_e \right), \tag{3}$$

where $|p'|$ is the number of functions in the execution path p' of G_f, p_s is a subpath (or segment) of p, $\delta(p_s)$ denotes the starting point of path p_s and $\psi(p_s)$ is the ending point of p_s.

E. Cost Model

The cost of executing a serverless application consists of the processing, transmission, and storage costs, which are defined as follows.

Processing cost: Serverless platforms adopt a new GB-second billing model that is determined by the memory size, the execution duration of each function, and the number of invocations [15], [23]. Note that the GB-second is a popular metric to capture the resource usages of serverless functions, which is defined as the number of seconds the function runs for multiplied by the amount of memory consumed. It is reasonable to measure how much resource the serverless functions consume, in relation to time and memory. For example, AWS Lambda charges 0.000016667 for every GB-second (per gigabyte per second) [15], [23]. Let c_l be the cost of using one GB of memory for one second to execute function f_l and c^{proc}_j the processing cost of implementing request r_j. Then, we have

$$c^{proc}_j = \sum_{f_l \in F} \sum_{CL_i \in \mathcal{CL}} (x_{j,l,i} + x'_{j,l,i}) \rho_j M_i D_i c_l N_l, \tag{4}$$

where D_i is the delay of processing a unit data in cloudlet CL_i.

Transmission cost: To implement a request r_j, its data needs to be transmitted to its application F in the MEC network. The transmission of states between a pair of functions incurs cost too, and this transmission cost is determined by the volume of transmitted data. Recall that $\mathcal{P}'$ is set of execution paths from the start point s_F to the end point t_F in the DAG G_f, and p' is one of such paths. The transmission cost of the original data and state of r_j is

$$c^{trans}_j = \sum_{\langle f_l, f'_l \rangle \in E_f} \sum_{p \in \mathcal{P}} \sum_{p_s \subseteq p} x_{j,l,\delta(p_s)} \cdot x_{j,l',\psi(p_s)} \sum_{e \in p_s} (\rho_j + a_{l,j}) \cdot c_e. \tag{5}$$

Storage cost: The storage cost is determined by the volume of state stored in a cloudlet. Let c^{sto}_j be the storage cost of executing functions by request r_j, then,

$$c^{sto}_j = \sum_{f_l \in F} \sum_{CL_i \in \mathcal{CL}} y_{j,l,i} \cdot a_{l,j} \cdot c_i, \tag{6}$$

where c_i is the cost of storing a unit volume of data in cloudlet CL_i and $y_{j,l,i}$ is to indicate whether the state of function f_l is stored in CL_i for r_j.

The total cost c_j of implementing request r_j in a serverless application F thus is the sum of its processing, storage and transmission costs, i.e.,

$$c_j = c^{proc}_j + c^{trans}_j + c^{sto}_j. \tag{7}$$

F. Formulations of Problems

We consider the following optimization problems.

Problem 1: Given a serverless application F and a set R of requests for F in an MEC $G = (\mathcal{CL}, \mathcal{E})$, assuming that the available resources in G are adequate for a serverless application F, *the stateful serverless application placement problem* in G is to jointly place a proper number instances (containers) of each function $f_l \in F$ to cloudlets of G and store the state of each function in the MEC, such that the total cost of implementing all requests in R is minimized while meeting the delay requirement of each request $r_j \in R$, subject to the memory resource capacity on each cloudlet. The problem can be mathematically formulated as

$$\mathbf{P1}: \quad \min \sum_{r_j \in R} c_j, \tag{8}$$

subject to the following constraints.

$$\sum_{CL_i \in \mathcal{CL}} (x_{j,l,i} + x'_{j,l,i}) = 1, \forall r_j \in R, \forall f_l \in F \tag{9}$$

$$\sum_{CL_i \in p} (x_{j,l,i} + x'_{j,l,i}) = z_{p',p}, \forall r_j, \forall p, \forall p', \forall f_l \in p' \tag{10}$$

$$\sum_{r_j \in R} \sum_{f_l \in F} (x_{j,l,i} + x'_{j,l,i}) M_i \cdot \rho_j \leq M(CL_i), \forall CL_i \tag{11}$$

$$x_{j,l,i} + x'_{j,l,i} \leq 1, \quad \forall r_j, \forall f_l, \text{ and } \forall CL_i \tag{12}$$

$$\sum_{CL_i \in \mathcal{CL}} y_{j,l,i} = 1, \quad \forall r_j, \forall f_l \tag{13}$$

$$\sum_{CL_i \in p} y_{j,l,i} = z_{p',p}, \ \forall r_j, \forall p, \forall p', \forall f_l \in p' \tag{14}$$

$$d_j \leq d_j^{req}, \quad \forall r_j \tag{15}$$

$$x_{j,l,i}, x'_{j,l,i}, y_{j,l,i}, z_{p',p}, \in \{0, 1\}, \tag{16}$$

where Constraint (9) says that each function f_l of request r_j has to be placed into a cloudlet CL_i. This indicates that any function f_l can only be placed into one single cloudlet and cannot be 'split' into multiple cloudlets. Constraint (10) shows that if the path p is assigned to an execution path p' in the DAG G_f of application F, each function in p' have to placed into a cloudlet in p. Constraint (11) ensures that the memory resource capacity of each cloudlet CL_i is met. Constraint (12) indicates that a function can be assigned to either an existing container or a newly-instantiated container. Constraint (13) makes sure that the state of each function is placed to a single cloudlet. Constraint (14) shows that if an execution path p' in DAG G_f is mapped to path p in the MEC network, each state of the functions in p' has to be placed into a cloudlet in p. Constraint (15) guarantees the delay requirement of each request not to be violated, and Constraint (16) ensures the integral constraint of each decision variable.

Problem 2: In reality, developers usually submit their functions of a serverless application F directly and the deployed functions implement user requests dynamically. The data volume of each request typically is unknown in advance, so are the transmission and processing delays. *The problem of dynamic stateful serverless application placement with the uncertainty of both data volume and delays* in an MEC network G is to jointly place the number of instances of each function f_l of serverless application F and store the state of each function, based on uncertain data volumes of requests and processing and transmission delays in G. As such, minimizing the total cost while meeting the delay requirements may not be achievable simultaneously, since there exists a non-trivial tradeoff between them. We instead aim to minimize a weighted sum of costs and delays of implementing all requests in R. The objective of the problem thus is to minimize

$$\mathbf{P2}: \quad \sum_{r_j \in R} (\xi \cdot c_j + (1 - \xi) \cdot d_j), \tag{17}$$

where ξ is the given weight on cost in the range of $(0, 1)$, subject to the constraints of (9) - (14), and (16).

Note that **P1** and **P2** are both integer cubic programs due to the fact that the delay of each request is determined by both the function placement and the path selection as shown in Eq. (3). Special versions of the problems without considering state dependency are basically virtual network embedding (VNE) problems that maps a virtual network (DAG) to a physical network (the MEC network) [40]. Since VNE problems are NP-Hard, the problems are NP-Hard as well. For clarity, the symbols used in this paper are summarized in Table I.

IV. Efficient Algorithms for the Stateful Serverless Application Placement Problem

In this section, we provide efficient solutions for the stateful serverless application placement problem (i.e., *Problem 1* defined in Section III-F). To this end, we first propose an efficient heuristic for the problem. Furthermore, we also develop an approximation algorithm with an approximation ratio for a special case of the problem without delay requirements.

A. Heuristic Algorithm

We jointly place a number of instances of each function f_l in F and store its state for later use by other functions. However, different functions in F may be distributed to different cloudlets, and they have dependencies among functions in F. Further, the state generated by one function may be demanded by the other functions in the same request as well as the other requests (request r_2 in Fig. 3). If the state is sent to every function of requests demanding it, the implementation cost of requests will be too high, as shown in Fig. 3 (a). To address this issue, the basic idea of the proposed algorithm is to introduce a *virtual state node* for the state of each function in G_f, which represents 'storing the state of each function in a cloudlet'. Each virtual state node is placed to a cloudlet that is close to the functions that demand the state, as shown in Fig. 3 (b). Specifically, a DAG G'_f containing virtual state nodes is constructed, referred as to the *state-placing DAG*. Then, both the original functions in G_f and the newly added state-placing functions in G'_f are placed to the MEC.

To reduce the processing delays, we find a parallel schedule of functions in G'_f by partitioning the state-placing DAG G'_f

TABLE I
SYMBOLS

Symbols	Meaning
$G = (\mathcal{CL}, \mathcal{E})$	an MEC network with a set $\mathcal{CL}$ of cloudlets and a set $\mathcal{E}$ of links between cloudlets.
CL_i and e	a cloudlet in $\mathcal{CL}$ and a link in $\mathcal{E}$.
$M(CL_i)$	the memory capacity of cloudlet CL_i.
d_e and c_e	the transmission delay and cost for transmitting a unit of data along link e.
R and r_j	the set of requests and a request in R.
ρ_j and v_j	the data volume generated by request r_j and the source node for r_j to access the MEC network.
F and f_l	a serverless application and a function in F.
N_l and $G_f = (V_f, E_f)$	the number of invocations of function f_l to implement a user request, and the DAG of a serverless application F.
$a_{l,j}$	the state generated by f_l during the processing of request r_j.
η_l	a proportional constant parameter in the range of (0,1).
$d^o_{l,i}$	the cold-start delay incurred by creating a container instance for function f_l in CL_i.
M_i	the amount of memory allocated by cloudlet CL_i to process a unit volume of data.
D_i	the delay for processing a unit of data in cloudlet CL_i.
α_i	an influence factor that captures cloudlet CL_i's impact of memory allocation on the processing delay.
$x_{j,l,j}$ and $x'_{j,l,j}$	the indicator variable shows whether a newly-instantiated container in cloudlet CL_i is used to implement function f_l of request r_j, and the indicator variable shows whether f_l of r_j is assigned to an existing container of cloudlet CL_i.
$d^{proc}_{l,j}(CL_i)$	the processing delay of request r_j by function f_l in cloudlet CL_i.
s_F and t_F	the entry function and exit function of an application F in the DAG G_f.
$\mathcal{P}'$ and p'	the set of execution paths from s_F to t_F in G_f and a path in $\mathcal{P}'$.
$\mathcal{P}$ and p	the set of paths in the MEC network to which the execution path p' in G_f is mapped and a path in $\mathcal{P}$.
$z_{p',p}$	the indicator variable that shows whether $p' \in \mathcal{P}'$ is mapped to $p \in \mathcal{P}$.
p_s, $\delta(p_s)$ and $\psi(p_s)$	the sub path of p, the starting point and the ending point of p_s.
d_j and d^{req}_j	the sum of processing and transmission delay of each request r_j and the delay requirement of each request r_j.
c_l and c_i	the cost of using one GB of memory for one second to execute f_l, and the cost of storing a unit volume of data in CL_i.
$c^{proc}_j(CL_i)$	the processing cost of implementing request r_j.
c^{trans}_j and c^{sto}_j	the transmission cost of implementing request r_j and the storage cost of r_j.
$y_{j,l,i}$	the indicator variable shows whether the state of function f_l is stored in cloudlet CL_i for request r_j.
c_j and ξ	the total cost of implementing a request r_j and the given weight on cost in the range of (0,1).
n_l	the virtual state node for function f_l that generates state.
G'_f and $G'(F)$	the state-placing DAG and the auxiliary graph.
$\mathcal{A}_F$ and $A_{F,q}$	the sequence of partition components for application F and qth partition component.
$CL_{i,q}$ and $CL'_{i,q}$	the virtual cloudlets for each $CL_i \in \mathcal{CL}$.
f^s_l and u_F	the function that generated state in partition component $A_{F,q-1}$, and a common sink node of the auxiliary graph $G'(F)$.
$\mu(\rho_j)$	the expected data volume of request r_j.
$\mu(D_i)$ and $\mu(d_e)$	the expected delay of processing a unit data in CL_i and transmitting a unit data along link e.
seq_o and $w_j(seq_o)$	a candidate cloudlet sequence can host functions in F and the current weight of seq_o.
$p_j(seq_o)$ and $pt_j(seq_o)$	the probability of choosing seq_o for request r_j and the penalty received by each cloudlet sequence seq_o.
ϵ	a given constant in the range of (0,1).

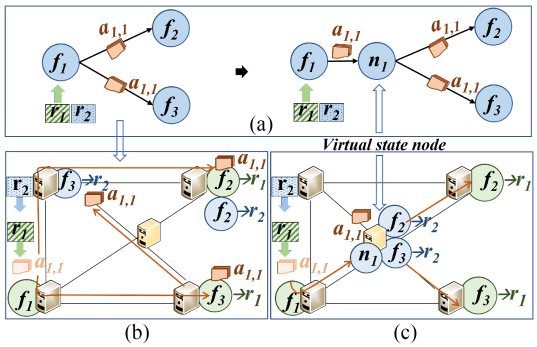


Fig. 3. A motivating example: (a) a serverless application with three functions f_1, f_2, and f_3, which are required by requests r_1 and r_2, and a serverless application with an introduced *virtual state node* n_1. Assume that request r_2 arrives later than r_1. The state $a_{1,1}$ generated by function f_1 of request r_1 is required not only by its functions but also functions of request r_2; (b) Fig. 3(b) shows sending $a_{1,1}$ to f_2 and f_3 of r_1 and r_2, separately, which incurs high transmission costs along long paths. (c) Instead, if state data $a_{1,1}$ is stored to a location that is close to both requests r_1 and r_2, via placing the virtual state node to the location as shown in Fig. 3(b), the transmission costs can be reduced significantly.

into a sequence of consecutive *partition components*. Each partition component has a maximal subset of functions that can be executed in parallel, and functions in a partition component can only be executed after the functions in its predecessor partition component have finished their executions. This means that the inputs of functions in a partition component in the sequence comes from the functions of its predecessor partition component and its outputs are forwarded to the functions of its successor partition component.

To reduce the total cost of placing functions of G'_f, we adopt a unified way of considering the processing, storage, and transmission costs, by assigning both edge and node costs in network G to the edges of another *auxiliary graph* $G'(F) = (V'(F), E'(F))$, which will be elaborated later. Built upon the state-placing DAG G'_f and auxiliary graph $G'(F)$, we then schedule requests in R for requesting serverless application F. The proposed algorithm consists of the following four stages.

Stage 1: Constructing a State-Placing DAG G'_f.

To place a duplicate of the state of a function f_l to a cloudlet, we add a virtual state node for each function f_l that generates state, denote by n_l the virtual state node. There is an edge $\langle f_l, n_l \rangle$ in the state placing DAG G'_f, meaning sending the state of f_l to node n_l for storage. Also, there is an edge from each virtual state node n_l to each function that needs the state. That is for each edge $\langle f_l, f_{l'} \rangle$ in the original DAG G_f, there is an edge $\langle n_l, f_{l'} \rangle$ in the state-placing DAG G'_f. Thus, placing each virtual state node n_l in G'_f means the placement of its corresponding state into a cloudlet in the MEC. Note that this scheme can be applied to the case where each state has multiple copies at different cloudlets. Along with all functions in G_f, their incoming/outgoing edges in G_f are added into G'_f too. An example of the state-placing DAG is shown in Fig. 4 (a).

Stage 2: Finding a Parallel Schedule.

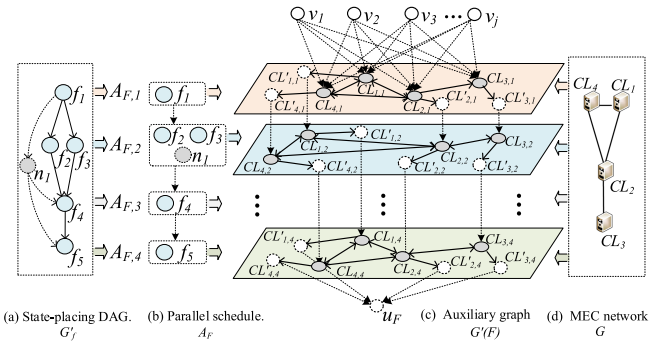


Fig. 4. An example of auxiliary graph $G'(F)$.

By applying the topological sort on the state-placing DAG G'_f [12], a schedule for the functions in G'_f can be found, which consists of a collection of sequential partition components. Each partition component contains a set of independent functions, as shown in Fig. 4 (b). Let $\mathcal{A}_F$ be the found sequence of partition components for application F, and $A_{F,q}$ be the qth partition component in the schedule. The partition components in $\mathcal{A}_F$ have disjoint sets of functions of G'_f. Since the functions in each partition component are consolidated into a single cloudlet, the output of functions in $A_{F,q}$ will be forwarded to the successor of the partition component (a.k.a. the found parallel schedule).

Stage 3: Building an Auxiliary Graph.

We now build an auxiliary graph $G'(F) = (V'(F), E'(F))$ based on the MEC $G = (\mathcal{CL}, \mathcal{E})$ for application F. Specifically, $G'(F)$ has $|\mathcal{A}_F|$ layers, each layer has the candidate cloudlets only for partition component $A_{F,q}$ of the state-placing DAG. Particularly, for each layer q of the rest layers, we first deal with the cloudlets in $\mathcal{CL}$. That is, for each $CL_i \in \mathcal{CL}$, we create two virtual cloudlets $CL_{i,q}$ and $CL'_{i,q}$. A directed edge from $CL_{i,q}$ to $CL'_{i,q}$ is added. The capacity of edge $\langle CL_{i,q}, CL'_{i,q}\rangle$ is set to $M(CL_i)$. The weight of edge $\langle CL_{i,q}, CL'_{i,q}\rangle$ is the cost of processing the original data volume of the functions of partition component $A_{F,q}$ and storing the states generated by functions in partition component $A_{F,q-1}$. Specifically, if there are states generated by functions in partition component $A_{F,q-1}$, the cost in processing and storing a unit volume of data in the edge from $CL_{i,q}$ to $CL'_{i,q}$ is

$$w\left(CL_{i,q}, CL'_{i,q}\right) = \sum_{f_l \in A_{F,q}} \left(M_i \cdot D_i \cdot c_l \cdot N_l\right) + \sum_{f_l^s \in A_{F,q-1}} \left(c_i \cdot \eta_l\right), \quad (18)$$

where f_l^s is the function that generated state in partition component $A_{F,q-1}$. Otherwise, we have

$$w(CL_{i,q}, CL'_{i,q}) = \sum_{f_l \in A_{F,q}} \left(M_i \cdot D_i \cdot c_l \cdot N_l\right). \quad (19)$$

The delay of edge $\langle CL_{i,q}, CL'_{i,q}\rangle$ in processing a unit data is set to $\max_{f_l \in A_{F,q}}(\alpha_i \cdot M_i \cdot N_l + d^o_{l,i})$. Note that such a delay is conservative, since the cold-start delay may be zero if some functions have already being instantiated.

For each original edge between CL_i and CL_{i+1} of the original network G, we add two directed edges, i.e., $\langle CL_{i,q}, CL_{i+1,q}\rangle$ and $\langle CL_{i+1,q}, CL_{i,q}\rangle$, into the auxliary graph $G'(F)$. We set the costs and delays of such edges in qth layer of $G'(F)$ to $\sum_{f_l^s \in A_{F,q-1}}(c_e \cdot (1+\eta_l))$ and $\sum_{f_l^s \in A_{F,q-1}}(d_e \cdot (1+\eta_l))$, if there are states generated by functions in partition component $A_{F,q-1}$; Otherwise, their costs and delays are set to c_e and d_e.

Recall that auxiliary graph $G'(F)$ has $|\mathcal{A}_F|$ layers with the qth layer having candidate cloudlets for partition component $A_{F,q}$. We then connect the nodes of $G'(F)$ layer by layer. That is, for each virtual cloudlet $CL'_{i,q}$, there is a directed edge from $CL'_{i,q}$ to $CL_{i,q+1}$. The cost and delay of this edge are zeros, as $CL'_{i,q}$ and $CL_{i,q+1}$ belong to the same cloudlet.

We add the source v_j of each request r_j of application F. We also add a common sink node u_F for all requests. There is an edge $\langle v_j, CL_{i,1}\rangle$ for r_j and each virtual cloudlet $CL_{i,1}$ in the first layer of $G'(F)$, and its cost and delays are set to zeros. There is an edge $\langle CL'_{i,|\mathcal{A}_F|}, u_F\rangle$ for each virtual cloudlet $CL'_{i,|\mathcal{A}_F|}$ in the last layer of $G'(F)$. An example of $G'(F)$ is shown in Fig. 4 (c).

Stage 4: Assigning Requests.

Note that each serverless application F needs to implement multiple requests in R. We then admit requests in R one by one, given the constructed state-placing DAG G'_f and auxiliary graph $G'(F)$ for application F. Specifically, we rank the requests in R in non-decreasing order of their data volume. For each sorted request r_j, we first prune some edges in auxiliary graph $G'(F)$ by removing the edges that cannot meet the memory resource demand of r_j. We then find a delay-constrained shortest path p_j in $G'(F)$ from source v_j to the virtual sink u_F. We finally find a feasible solution to the stateful serverless application placement problem, which is derived from p_j. That is, for each layer q, if edge $\langle CL'_{i,q}, CL_{i,q+1}\rangle$ is contained in path p_j, there will be an instance of partition component $A_{F,q}$ in cloudlet CL_i. We then update the residual memory capacity of each cloudlet in the auxiliary graph and assign the next sorted request. This procedure continues until all requests are admitted.

The detailed algorithm for the problem is given **Algorithm 1**, or `Heu` for short cut.

B. Approximation Algorithm for a Special Case of the Problem Without Delay Requirements

We now consider a special case of the problem, for which we propose an approximation algorithm. The proposed approximation algorithm is almost identical to algorithm `Heu`, and the only difference lies in its last stage that is described in the following. Given the state-placing DAG G'_f and auxiliary graph $G'(F)$, we reduce the problem without delay requirements in G to a minimum-cost multicommodity flow problem in an auxiliary graph, where we treat each request r_j as a commodity with source v_j and a destination u_F, and its demand is the data volume ρ_j of r_j. We then find an unsplittable minimum multicommodity flow for the $|R|$ commodities in $G'(F)$. A feasible flow corresponds to the placement of partition components of the state-placing DAG G'_f, which corresponds to the function scheduling of a serverless application F. That is, for each layer

Algorithm 1: Heu.

Input: $G = (\mathcal{CL}, \mathcal{E})$, a serverless application F, a set R of requests, and their delay requirements.
Output: An assignment of each request $r_j \in R$ and a number of placed instances in cloudlets for each function $f_l \in F$.
1: For application F, construct the state-placing DAG G'_f, as shown in Fig. 4 (a);
2: Build an auxiliary graph $G'(F)$ based on the MEC, which consists of $|\mathcal{A}_F|$ layers, each layer is constructed according to MEC G, as shown in Fig. 4 (c);
3: Rank the requests in R into an increasing order of their data volumes;
4: **for** each request r_j of the ranked list **do**
5: Remove the edges that cannot meet the memory resource demand of r_j;
6: Find a constraint shortest path p_j from v_j to u_F in auxiliary graph $G'(F)$;
7: **for** Each partition component $A_{F,q}$ **do**
8: **if** $\langle CL'_{i,q}, CL_{i,q+1}\rangle$ is in path p_j **then**
9: Create an instance for each function or virtual state node in partition component $A_{F,q}$ in cloudlet CL_i.

Algorithm 2: Appro.

Input: $G = (\mathcal{CL}, \mathcal{E})$, a serverless application F and a set R of requests.
Output: An assignment of each request $r_j \in R$ and a number of placed instances in cloudlets for each function $f_l \in F$ and state $a_{l,j}$ of f_l.
1: Build a state-placing DAG G'_f and auxiliary graph $G'(F)$ following the first three stages of algorithm Heu;
2: Consider each request r_j as a commodity with source v_j, destination u_F and demand ρ_j;
3: Find a unsplittable flow f for the $|R|$ commodities in the auxiliary graph $G'(F)$;
4: **for** each partition component $A_{F,q}$ **do**
5: **if** $\langle CL'_{i,q}, CL_{i,q+1}\rangle$ is in flow f **then**
6: Create an instance for each function or virtual state node in partition component $A_{F,q}$ in cloudlet CL_i.

q, if the flow along edge $\langle CL'_{i,q}, CL_{i,q+1}\rangle$ is non-negative, the partition component $A_{F,q}$ will be placed to cloudlet CL_i.

The detailed steps of **Algorithm** Appro are given below.

C. Algorithm Analysis

In the following, we first analyze the solution feasibility and time complexity of Heu. We then analyze the approximation ratio of the proposed approximation algorithm.

Theorem 1. Algorithm Heu delivers a feasible solution to the stateful serverless application placement problem in time $O(|R|\log|R| + |R|(|E_f||V_f|(\log\log|V_f|)))$.

Proof. We show that the solution of algorithm Heu is feasible, by showing (1) the memory constraint on each cloudlet is met; (2) the delay requirement of each request is met; and (3) the states are transferred to the functions that demand them accordingly. For the memory constraint, we can see that in the construction of auxiliary graph $G'(F)$, the capacity of each cloudlet CL_i is assigned to virtual edge $\langle CL_{i,q}, CL'_{i,q}\rangle$. Also, in **Stage 4**, we remove the edges that cannot meet the memory resource demand of a request. Thus, the memory capacity of each cloudlet is met. For the delay requirement, we find a delay-constraint shortest path in the constructed auxiliary graph $G'(F)$. The delay requirement of each request is satisfied. For the state dependency constraint, although we add a virtual state node for each edge $\langle f_l, f_{l'}\rangle$, the constraint is met in the state-placing DAG G'_f.

The rest is to analyze the running time of algorithm Heu. In **Stage 1**, we construct the state-placing DAG G'_f based on the DAG G_f with both function and state dependencies, by adding a virtual state node for each function f_l if it generates state. Considering that not all functions in F generate states, the complexity of constructing G'_f is $O(|V_f|^2)$. In **Stage 2**, we find a topological sort in G'_f, which takes $O(|V_f| + |E_f|)$ time [12]. The most time-consuming part of **Stage 3** is due to finding a delay constrained shortest path for each request. The time complexity thus is $O(|R|\log|R| + |R| \cdot (|E_f||V_f|(\log\log|V_f|)))$, where the time due to finding the delay constrained shortest path is $O(|E_f||V_f|(\log\log|V_f|))$ [24]. In **Stage 4**, it takes $O(|\mathcal{CL}|)$ time to derive a feasible solution for the original problem from the delay constraint shortest path for request r_j. The time complexity of Heu thus is $O(|R|\log|R| + |R| \cdot (|E_f||V_f|(\log\log|V_f|)))$.

We then analyze the performance of algorithm Appro. □

Lemma 1. The violation on the memory capacity of each cloudlet by algorithm Appro is upper bounded by a ratio of $\frac{\rho_{\max}(|\mathcal{A}_F|-1)}{\rho_{\min}\cdot M_{\max}}$, where $M_{\max} = \max\{M(CL_i)\}$, $\rho_{\max} = \max\{\rho_j\}$, $\rho_{\min} = \min\{\rho_j\}$, $\mathcal{A}_F$ is the sequence of partition components for application F, and $|\mathcal{A}_F|$ is the number of partition components in F.

Proof. According to the construction of auxiliary graph $G'(F)$, functions in application F may be assigned to the same cloudlet CL_i. However, edges related to CL_i in the auxiliary graph, such as the edge $\langle CL_{i,q}, CL'_{i,q}\rangle$ in each layer q, have the same capacity. In the worse case, a cloudlet is saturated in the first layer of the auxiliary graph, by implementing a request with the minimum data volume $\rho_{\min}$. In each of the rest layers, the cloudlet is assigned to the request with the maximum data volume $\rho_{\max}$. This means that the violation ratio is $\rho_{\max}(|\mathcal{A}_F| - 1)/(\rho_{\min} \cdot M_{\max})$. □

Theorem 2. The approximation ratio of algorithm Appro is $\max\{\gamma/c_{e,min}, |\mathcal{CL}|\}\sigma(a_{\min}/\rho_{\max})$, where $\sigma = (\rho_{\max} \cdot c_{\max})/(\rho_{\min} \cdot c_{\min})$, $\rho_{\max} = \max\{\rho_j\}$, $\rho_{\min} = \min\{\rho_j\}$, $c_{\max} = \max\{c_l\}$, $c_{\min} = \min\{c_l\}$, $c_{e,min} = \min\{c_e\}$, and $a_{\min} = \min\{a_{l,j}\}$.

Proof. Let OPT be the optimal solution to the stateful serverless application placement problem in G, and OPT' be the optimal solution to the minimum-cost multicommodity flow problem in $G'(F)$. We have $OPT \leq OPT'$ because placing an instance of a partition component $A_{F,q}$ to a single cloudlet pushes up the costs. Let C be the obtained solution by Appro. Kolliopoulos and Stein [20] showed that the approximation ratio of a minimum cost flow can be imporved to a constant arbitrarily close to 1 at the expense of an increase in the performance guarantee for link congestion, i.e., $C/OPT' \approx 1$.

Algorithm Appro first constructs a state-placing DAG G'_f by adding a virtual state node to denote the placement of a state to a cloudlet in the MEC. It then adopts a topological sort in G'_f to obtain a sequence of partition components with each partition component containing a set of independent functions. Each partition component is assigned to a single cloudlet. However, the optimal solution OPT may place the entire application F in the granularity of functions, with each function being placed to the cloudlet with the minimum cost. While, in the worst case, algorithm Appro places each partition component to a cloudlet with the maximum sum of the costs of processing, transmission and storage. This means that the total cost can be increased by a ratio of σ in the worst case. Also, let n_f be the number of instances that OPT places for each function f_l in application F, and denote by n'_f such a number by algorithm Appro. If $n_f > n'_f$, it means that the optimal solution prefers to reduce the transmission cost by placing more instances of each function, making them close to each other. The cost of Appro then can be increased by at most $\rho_{\max} \cdot O(|E|)/c_{e,min}$ times, where $O(|E|)$ represents a upper bound of the length of a path in the MEC and $c_{e,min} = \min\{c_e\}$. Otherwise, processing and storage costs dominate the total cost. In particular, the processing cost can be pushed up by at most a factor of $\rho_{\max}|\mathcal{CL}|$. Since the storage cost is counted according to the data volume ρ_j of each request instead of the volume of its state, this means that the cost of OPT' may push up the optimal solution OPT at most by $\rho_{\max}/a_{\min}$, where $a_{\min} = \min\{a_{l,j}\}$, i.e.,

$$OPT' > (\rho_{\max}/a_{\min})\, OPT. \tag{20}$$

We thus have

$$\begin{aligned}\frac{C}{OPT} &\leq \max\left\{\frac{O(|E|)}{c_{e,min}}, |\mathcal{CL}|\right\} \cdot \sigma \cdot C \cdot (a_{\min}/\rho_{\max})/OPT' \\ &\leq \max\left\{O(|E|), |\mathcal{CL}|\right\} \cdot \sigma \cdot (a_{\min}/\rho_{\max}). \\ &= O(|E|) \cdot \sigma \cdot (a_{\min}/\rho_{\max}). \end{aligned} \tag{21}$$

In particular, when the gap between the maximum and minimum costs of transmitting the data of each request along an edge is larger, σ is larger. Consequently, the approximation ratio may be worse (larger). The reason is that more functions of each parallel partition of the DAG may not be placed into locations with optimal costs. □

V. An Online Learning Algorithm for the Stateful Serverless Application Placement Problem With the Uncertainty of Both Data Volumes and Network Delays

In this section, we devise an online learning algorithm with a bounded regret for the stateful serverless application placement problem with the uncertainty of both data volumes and delays (i.e., *Problem 2* defined in Section III-F).

A. Basic Idea

To predict the data volume of each request and network delays precisely, an online learning algorithm is desperately needed. Assume that the expected data volume of each request and the processing and transmission delays are given. Let $\mu(\rho_j)$ be the expected data volume of request r_j, and let $\mu(D_i)$ be the expected delay of processing a unit data in cloudlet CL_i and $\mu(d_e)$ the expected delay of transmitting a unit data along link e. Our basic idea is to find a set of candidate placements for functions in a serverless application according to available statistical information on data volumes and delays. Each candidate placement is treated as an 'arm' in a multi-armed bandit game. The online learning algorithm based on multi-armed bandit technique then evaluates the loss of each arm and adjusts the probability of each arm choice dynamically.

B. An Online Learning Algorithm

Given the expected data volume $\mu(\rho_j)$ of request r_j, the expected processing delay $\mu(D_i)$, and the expected transmission delay $\mu(d_e)$, the first two stages of our algorithm are identical to those in Appro. In the third stage, the only difference lies in the edge weight settings. We construct a similar auxiliary graph as shown in Fig. 4 (c), denoted by $G'(F)$. If there are states generated by functions in partition component $A_{F,q-1}$, the cost of edge $\langle CL_{i,q}, CL'_{i,q}\rangle$ for processing and storing a unit volume of data is

$$\begin{aligned} w\left(CL_{i,q}, CL'_{i,q}\right) = \xi \cdot \mu \Bigg(&\sum_{f_l \in A_{F,q}} (M_i \cdot D_i \cdot c_l \cdot N_l) \\ &+ \sum_{f_l^s \in A_{F,q-1}} (c_i \cdot \eta_l) \Bigg) + (1-\xi) \\ &\mu\left(\max_{f_l \in A_{F,q}} \left(\alpha_i \cdot M_i \cdot N_l + d^o_{l,i}\right)\right); \end{aligned} \tag{22}$$

otherwise, we have

$$\begin{aligned} w\left(CL_{i,q}, CL'_{i,q}\right) = \xi \cdot \mu &\left(\sum_{f_l \in A_{F,q}} (M_i \cdot D_i \cdot c_l \cdot N_l)\right) \\ + (1-\xi) \cdot \mu &\left(\max_{f_l \in A_{F,q}} \left(\alpha_i \cdot M_i \cdot N_l + d^o_{l,i}\right)\right). \end{aligned} \tag{23}$$

The costs of edges $\langle CL_{i,q}, CL_{i+1,q}\rangle$ and $\langle CL_{i+1,q}, CL_{i,q}\rangle$ in the qth layer are set to

$$\begin{aligned} &\xi \sum_{f_l^s \in A_{F,q-1}} (c_e \cdot (1+\eta_l)) \\ &\quad + (1-\xi)\mu \left(\sum_{f_l^s \in A_{F,q-1}} (d_e \cdot (1+\eta_l))\right), \end{aligned} \tag{24}$$

if there are states generated by functions in partition component $A_{F,q-1}$; otherwise, their costs are

$$\xi \cdot c_e + (1-\xi) \cdot \mu(d_e). \tag{25}$$

In the fourth stage, we find a splittable minimum-cost multicommodity flow f'. We then determine candidate cloudlet sequences of implementing requests. Specifically, let p_o be one of the paths in the splittable flow f' from a source to the common

destination u_F in the auxiliary graph $G'(F)$. That is, for each edge $\langle CL_{i,q}, CL'_{i,q}\rangle$ in path p_o, we assume that cloudlet CL_i is considered as a cloudlet to implement the functions in the qth partition component of the state-placing DAG G'_f. Thus, all cloudlets in path p_o compose of a candidate cloudlet sequence. A candidate cloudlet sequence can host functions of F and process a non-negative volume of data flow of a request. For clarity, we use seq_o to represent such a candidate cloudlet sequence, and the qth cloudlet in the sequence hosts functions in qth partition component.

Notice that the candidate cloudlet sequences are selected based on statistical information of requests and the MEC network. Implementing requests in such cloudlet sequences may violate the capacity constraints and delay requirements. To avoid such violations, we then perform a dynamic adjustment procedure based on the technique of multi-armed bandits. Specifically, we consider each candidate cloudlet sequence as an arm. On the arrival of request r_j, an arm needs to be selected to implement function instances of the application to implement request r_j. Initially, the algorithm assigns each arm a weight of 1. As the algorithm proceeds, we degrade the weight of each arm (cloudlet sequence) according to the cost and delay violations received due to selecting the corresponding cloudlet sequence to host function instances of the application.

Based on the weight of each cloudlet sequence, we calculate the probability of selecting it as the cloudlet to place function instances of the application F for request r_j. That is, we select a cloudlet sequence seq_o for request r_j with a probability that is proportional to its weight. Denote by $w_j(seq_o)$ the current weight of sequence seq_o, and $w_j(seq_{o'})$ the current weight of the remaining candidate sequence in $G'(F)$ except seq_o before the admission of r_j, and $p_j(seq_o)$ the probability of choosing seq_o for request r_j, then,

$$p_j(seq_o) = w_j(seq_o)/\sum_{o'} w_j(seq_{o'}). \tag{26}$$

If the cloudlet sequence seq_o is selected and there are already instantiated functions in each cloudlet of seq_o, request r_j will be assigned to the instantiated functions. Otherwise, for the qth cloudlet in the sequence, we create an instance of the qth partition component of the state-placing DAG G'_f. Once request r_j is implemented, its data volume will be revealed and the processing and transmission delays will be known. Likewise, the cost of implementing the request and delay violations can be obtained. We then adjust the probabilities of selecting a cloudlet sequence, according to the cost loss and delay violations. To minimize the cost and avoid delay violations, we define the penalty received by each cloudlet sequence seq_o as a weighted sum of cost and delay for implementing request r_j in the sequence, i.e.,

$$pt_j(seq_o) = \xi \cdot c_j + (1-\xi)\cdot d_j, \tag{27}$$

where c_j and d_j are the cost and delay of implementing request r_j. Intuitively, if a cloudlet sequence seq_o generates a higher cost with more delay violations, we decrease its weight after implementing request r_j by

$$w_{j+1}(seq_o) = w_j(seq_o)\cdot(1-\epsilon)^{pt_j(seq_o)}, \tag{28}$$

Algorithm 3: OL.

Input: $G=(\mathcal{CL},\mathcal{E})$, a serverless application F, a set R of requests.

Output: An assignment of each request $r_j \in R$ and a number of placed instances in cloudlets for each function $f_l \in F$.

1: Execute the first two stages of algorithm Appro;
2: Build an auxiliary graph $G'(F)$ according to Fig. 4, and set its edge costs and delays according to equations (22), (23), (24), and (25);
3: Find a splittable minimum-cost multicommodity flow in the auxiliary graph, and let f' be the obtained flow;
4: Obtain a set of candidate cloudlet sequences from f';
5: **for** each arrived request $r_j \in R$ **do**
6: Calculate the probability $p_j(seq_o)$ of selecting a cloudlet sequence seq_o to implement r_j by Eq. (26);
7: Select the seq_o with probability $p_j(seq_o)$;
8: Each selected cloudlet sequence either creates new function instances of application, or assigns r_j to an already created instance;
9: Observe the data volumes and delays experienced by request r_j, and calculate the obtained penalty;

where ϵ is a given constant in the range of $(0,1)$. The above procedure continues until each request is implemented.

Note that the proposed algorithm can be applied directly to consider the uncertainty of execution flows of an application. Specifically, if some functions in a partition component are not executed, the cloudlets that execute the component may experience a sudden change in execution delays. This can be quickly learnt by the proposed algorithm, because each sequence of cloudlets executing the parallel schedule is assigned to a dynamically-changing weight. As such, if such cases happen, the weight can be decreased if the mis-configuration caused higher delays. The detailed steps are shown in *Algorithm* OL.

C. Regret Analysis

Theorem 3. Algorithm OL has a regret of $\frac{\ln|R|}{\epsilon} + 2\epsilon PT_d$, assuming that there is a bound on the minimum and maximum objective values of each commodity, let PT_d be this bound, i.e., $PT_d = \max_j(\xi\cdot c_j + (1-\xi)\cdot d_j) - \min_j(\xi\cdot c_j + (1-\xi)\cdot d_j)$, and $\epsilon \in (0, 1/2)$.

Proof. We first define the regret of selecting candidate cloudlet sequences obtained by algorithm OL. Algorithm OL makes decisions based on the uncertainty of both data volume and delays, which may push up the cost or violate the delay requirements. Intuitively, the algorithm expects to reduce the cost and delay violation jointly. We thus define the regret of an online learning algorithm as the expected penalty deviation of the weighted sum of the obtained solution from the optimal one, i.e.,

$$Reg(R) = pt(\texttt{OL}) - pt^*, \tag{29}$$

where $pt(\texttt{OL})$ is the penalty of algorithm OL for implementing requests in R and pt^* is the penalty obtained by the best candidate

cloudlet sequence with the lowest cost, which satisfies the delay and memory capacity constraints.

We use W_j to denote the total weight of all cloudlet sequences before implementing the currently arrived request r_j, i.e., $W_j = \sum_o w_j(seq_o)$. Then, the weight of each cloudlet sequence after implementing the last request $r_{|R|}$ is

$$w_{|R|+1}(seq_o) = w_1(seq_o)\Pi_{j=1}^{|R|}(1-\epsilon)^{pt_j(seq_o)}.$$

Similarly, the total weight of all cloudlet sequences after implementing the last request is

$$W_{|R|+1} > w_{|R|+1}(seq^*) = (1-\epsilon)^{pt^*} > (1-\epsilon)^{pt_{|R|}(seq_o)}.$$

Due to the definition of costs and the fact that $(1-\epsilon)^a > (1-\epsilon)^b$ for any positive values of a and b with $1 < a < b$.

We then have

$$\begin{aligned} W_{j+1}/W_j &= \left(\sum_o w_{j+1}(seq_o)\right)/W_j \\ &= \left(\sum_o (1-\epsilon)^{pt_j(seq_o)} \cdot w_j(seq_o)\right)/W_j, \text{due to Eq. (28)} \qquad (30) \\ &< \sum_o \left(1-\alpha\cdot pt_j(seq_o)+\beta\cdot pt_j(seq_o)^2\right)\cdot p_j(seq_o), \\ &\text{since } W_j \geq 1 \\ &\text{and } \exists \alpha,\beta>0, (1-\epsilon)^x < 1-\alpha\cdot x+\beta\cdot x^2 \text{ for all } x>0. \\ &= \sum_o p_j(seq_o) - \alpha\sum_o p_j(seq_o)\cdot pt_j(seq_o) \\ &\quad + \beta\sum_o p_j(seq_o)\cdot pt_j(seq_o)^2 \\ &= 1-\alpha\sum_o E(pt_j(seq_o)) + \beta\sum_o \left(E\left(pt_j(seq_o)\right)\right)^2 \\ &= 1-\alpha E\left(pt_j(\texttt{OL})\right)+\beta\left(E\left(pt_j(\texttt{OL})\right)\right)^2, \qquad (31) \end{aligned}$$

where $pt_j(\texttt{OL})$ is the total penalty of all cloudlet sequences after implementing request r_j. Clearly, $pt(\texttt{OL}) = \sum_{j=1}^{|R|} pt_j(\texttt{OL})$. By taking a logarithm on inequality (31), we have

$$\begin{aligned} &\ln\left(W_{j+1}/W_j\right) \\ &\quad < \ln\left(1-\alpha\cdot E(pt_j(\texttt{OL}))+\beta\cdot E(pt(\texttt{OL})^2)\right) \\ &\quad < -\alpha\cdot E\left(pt_j(\texttt{OL})\right)+\beta\cdot E(pt_j(\texttt{OL})^2), \qquad (32) \end{aligned}$$

since $\ln(1-x) < -x$ for any $x \in (0,1)$. In particular, this holds when $(\alpha,\beta) = (\epsilon, 0)$.

Considering all requests in R with $1 \leq j \leq |R|$, we have

$$\begin{aligned} &\sum_{j=1}^{|R|}\left(\alpha\cdot E(pt_j(\texttt{OL}))-\beta\cdot E\left(pt_j(\texttt{OL})^2\right)\right) \\ &\quad < -\ln(W_{j+1}/W_j) < -\ln\Pi_{j=1}^{|R|}(W_{j+1}/W_j) \\ &\quad = -\ln\left(W_{j+1}/W_j\right) = \ln W_1 - \ln W_{|R|+1} \\ &\quad < \ln|R| - \ln(1-\epsilon)pt^*. \qquad (33) \end{aligned}$$

With $(\alpha,\beta) = (\epsilon, 0)$, we have $E(pt(\texttt{OL})) < \ln|R|/\epsilon + (1/\epsilon)\ln[1/((1-\epsilon))]E(pt^*)$. Assuming that $\epsilon \in (0, 1/2)$, we have $\frac{1}{\epsilon}\ln\frac{1}{(1-\epsilon)} \leq 1+2\epsilon$, which means

$$E\left(pt(\texttt{OL})-pt^*\right) < \ln|R|/\epsilon + 2\epsilon E\left(pt^*\right). \qquad (34)$$

Let $PT_d = \max_j(\xi\cdot c_j+(1-\xi)\cdot d_j) - \min_j(\xi\cdot c_j+(1-\xi)\cdot d_j)$, we have $pt^* < PT_d$. The regret of algorithm OL thus is $E(pt(\texttt{OL})-pt^*) < \ln|R|/\epsilon + 2\epsilon\cdot PT_d$. □

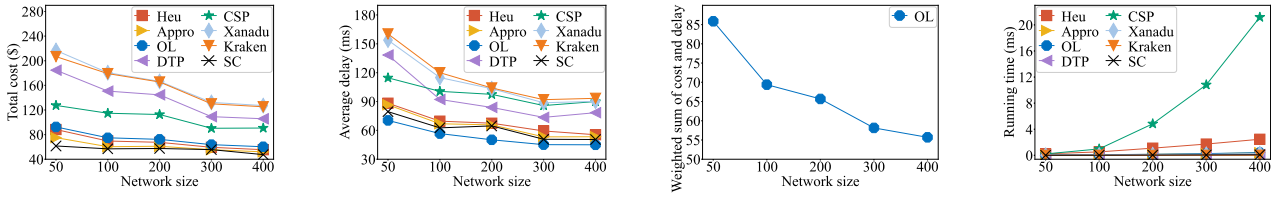
VI. Performance Evaluation

We now evaluate the performance of the proposed algorithms against existing studies by extensive simulations.

A. Parameter Settings

We consider an MEC network consisting of cloudlets with size varying from 50 to 400 nodes, where each network topology is generated using GT-ITM [16]. The number of cloudlets in each network is set to 10 percent of the network size. The memory capacity of each cloudlet varies from 5 to 8 GB [37]. The amount of memory required for each cloudlet to process a unit volume of data is within [10, 30] Mega Bytes (MBs), and the correspondingly incurred delay is set within [0.05, 0.09] milliseconds [32]. We deploy functions on AWS Lambda, which allows the allocated memory to vary between 64 MB and 3,008 MB in 64 MB increments [23]. Therefore, we set the memory allocated to each function based on the value to the maximum memory required by the cloudlet to process data volume per request. For example, if a cloudlet requires a maximum of 100 MB of memory to process data volume per request, we allocate memory to be the closest allowed value by AWS Lambda which is 128 MB. We employ AWS Step Functions to create a workflow of lambda functions [2], which is a serverless workow coordination service that combines multiple Lambda functions and other serverless services offered by AWS into responsive serverless applications. The number of invocations of each function to implement a user request varies from 1 to 5 times, and the cost of using one GB of memory for one second to execute each function follows AWS Lambda's current pricing, which depends on the amount of memory allocated to each function [1]. The cost of using a unit amount of storage resource in a cloudlet is set within [$0.005, $0.015]. The transmission cost and transmission delay for transmitting a unit volume of data along link e are set within [$0.0001, $0.0005] and [0.01, 0.05] milliseconds [38], respectively. The data volume of each request is randomly drawn from [50, 100] MB/s, and the delay requirement is randomly generated from [30, 50] milliseconds. Unless otherwise specified, we will adopt the default settings in our experiments.

We evaluated our algorithms against the following benchmarks: (1) a workflow-aware resource management framework (referred to as Kraken) that minimizes the number of containers provisioned for an application DAG while ensuring SLO requirements [8]; (2) a Dynamic Task Placement mechanism (referred to as DTP): which dynamically selects cloud or edge pipelines to execute functions based on workload characteristics and delay requirements [13]; (3) a proactive deployment mechanism



(a) Total cost. (b) Average delay. (c) Weighted sum of cost and delay. (d) Running time.

Fig. 5. The impact of the network size on the performance of algorithms Heu, Appro, OL, DTP, CSP, Xanadu, Kraken and SC.

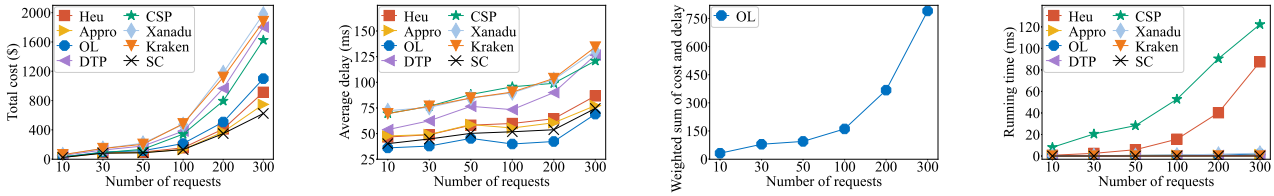
(referred to as Xanadu) that eliminates the cold-starts overhead by detecting a most-likely-path in a function workflow and pre-provisioned the execution sandboxes [14]; (4) a Constrained Shortest Path mechanism (referred to as CSP): each function can be placed by fusing a sequence of functions across edge and cloud resources [15]; and (5) an algorithm that places all functions of a serverless application in a single cloudlet without considering the memory capacity constraint of each cloudlet, which is referred to as SC. It must be mentioned that why we select the above benchmark algorithms is due to the reasons that (1) both of them study the problem of serverless function placement. Specifically, studies in [8], [14], [15] define the workflow of an application as a DAG, and [13] considers the storage of the execution results of functions; and (2) studies in [13] and [15] both incorporate processing and transmission costs with the aim to minimize the total cost of functions; and (3) studies in [8] and [14] both investigate the resource provisioning problem to reduce delay.

B. Performance Evaluation

We first evaluated the performance of algorithms Heu, Appro and OL against that of DTP, CSP, Xanadu, Kraken and SC in terms of the total cost and average delay, by varying the network size from 50 to 400 while fixing the number of requests at 30. From Fig. 5 (a) and (b), we can see that Appro, Heu and OL achieve lower costs with less average delays compared to algorithms DTP, CSP, Xanadu and Kraken. The reason is two fold: (1) Appro, Heu and OL adopt an efficient application placement strategy that considers function and state dependencies, such that the costs due to state processing and transmission are reduced; and (2) Appro and OL adopt a global optimization strategy based on the minimum-cost multicommodity flow algorithm. However, algorithm SC achieves the lowest cost because all functions of a serverless application are placed in a single cloudlet, which avoids transmission cost and delay. Further, we can see that the total costs and average delays of algorithms Heu, Appro, OL, DTP, CSP, Xanadu, Kraken and SC decrease with the increase on network size. The rationale behind is that with the growth of network size, each function has more opportunities to be executed in cloudlets with lower processing cost and larger memory capacity, which also means that each request can be implemented in cloudlets that are very close to its home cloudlet, thus reducing transmission and processing delays. It must be mentioned that Heu, Appro and OL still have good scalability when applied to larger network sizes. Since the objective of algorithm OL is to minimize a weighted sum of cost and delay of implementing requests, it performs better in terms of the average delay. The weighted sum of cost and delay obtained by OL is shown in Fig. 5 (c). The running time of CSP and Heu increase with the growth of network size due to the greedy search for the optimal path. In contrast, Appro, OL, DTP, Xanadu, Kraken and SC show better running times, as shown in Fig. 5 (d).

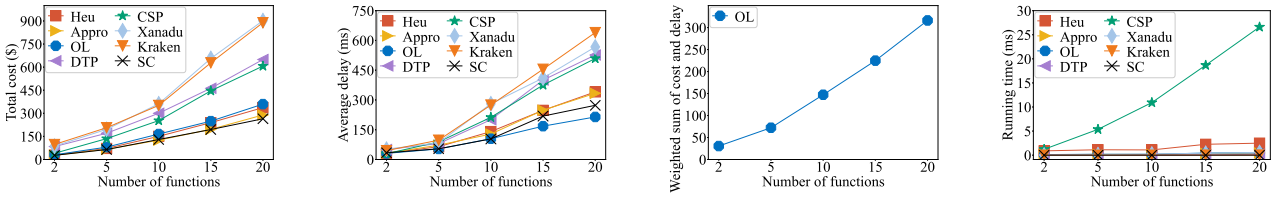
We then investigated the impact of the number of requests on the performance of algorithms Heu, Appro, OL, DTP, CSP, Xanadu, Kraken and SC, by varying the number of requests from 10 to 300 while fixing the network size at 400. As can be seen from Fig. 6 (a), algorithms Heu, Appro and OL achieve lower total costs than these of algorithms DTP, CSP, Xanadu and Kraken, which means that the proposed algorithms can still maintain a good applicability to large-scale application processing requests. Also, SC still shows better performance in terms of total cost and average delay, because it is not limited by the memory resource capacity of each cloudlet. Further, the total cost increases with the growth of the number of requests. The rationale behind is that when the number of requests increases, more memory resources are needed to execute the functions demanded by requests. This increases the cost of leasing the memory resources. Also, the average delay may be increased as more requests are implemented in cloudlets with long processing delays due to their memory capacity constraints, as shown in Fig. 6 (b). A similar trend for the weighted sum of cost and delay of OL is shown in Fig. 6 (c) by setting ξ to 0.7. The growth in the number of requests that need to be implemented increases the running time, as shown in Fig. 6 (d).

We also studied the impact of the number of functions of each application on the performance of algorithms Heu, Appro, OL, DTP, CSP, Xanadu, Kraken and SC, by varying the number of functions from 2 to 20. From Fig. 7 (a), (b), (c) and (d), we can see that the total costs, average delays, weighted sum of cost and delay, and running time of algorithms Heu, Appro, OL, DTP, CSP, Xanadu, Kraken and SC increase when the number of functions grows from 2 to 20. This is because that more resources are needed to execute the increasing number of functions for each request. Meanwhile, due to memory resource constraints, this may cause the data to be transferred between multiple cloudlets for each application. In addition, Heu, Appro and OL achieve lower total costs compared with those of DTP, CSP, Xanadu and Kraken. This is because that CSP, DTP neglect the reuse of states generated by functions, thereby pushing up the storage



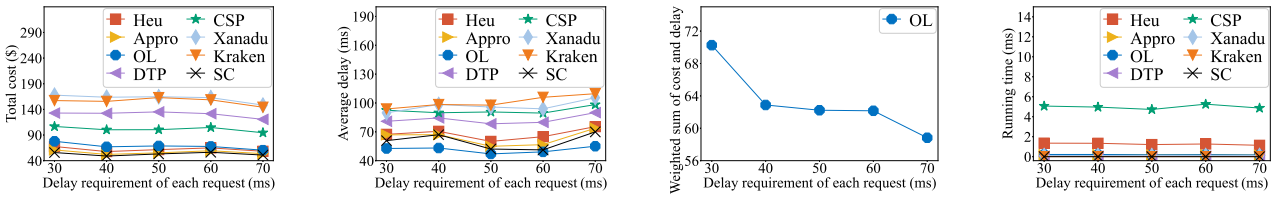
(a) Total cost. (b) Average delay. (c) Weighted sum of cost and delay. (d) Running time.

Fig. 6. The impact of the number of requests on the performance of algorithms Heu, Appro, OL, DTP, CSP, Xanadu, Kraken and SC.



(a) Total cost. (b) Average delay. (c) Weighted sum of cost and delay. (d) Running time.

Fig. 7. The impact of the number of functions on the performance of algorithms Heu, Appro, OL, DTP, CSP, Xanadu, Kraken and SC.



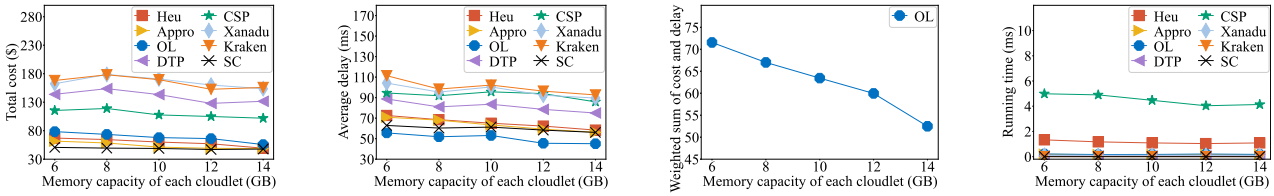
(a) Total cost. (b) Average delay. (c) Weighted sum of cost and delay. (d) Running time.

Fig. 8. The impact of the delay requirement of each request on the performance of algorithms Heu, Appro, OL, DTP, CSP, Xanadu, Kraken and SC.

costs, and Xanadu and Kraken do not jointly optimize the computing and transmission costs.

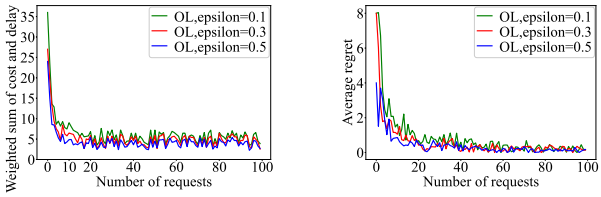
To demonstrate the impact of delay requirement of each request on the performance of Heu, Appro, OL, DTP, CSP, Xanadu, Kraken and SC, we vary the delay requirement of each request from 30 ms to 70 ms while fixing the network size at 200. From the Fig. 8 (a), we can see that the total costs of algorithms Heu, Appro, OL, DTP, CSP, Xanadu, Kraken and SC decrease as the delay requirements of requests are gradually relaxed. This is because functions can be executed at cloudlets with lower processing costs by sacrificing a certain amount of delay. In contrast, the average delay increases with the growth of delay requirement of each request. The rationale behind is that each request may be implemented in cloudlets over a long distance or with a higher processing delay, which undoubtedly increases the transmission delay under the goal of minimizing cost, as shown in Fig. 8 (b). By setting ξ at 0.7, the weighted sum of cost and delay obtained by algorithm OL is shown in Fig. 8 (c). We can see from the Fig. 8 (d) that the algorithms show a steady trend in the running time of implementing requests, since the increased delay requirement of each request has little effect on the running time of the algorithms.

We further evaluated the impact of the memory capacity of each cloudlet on the performance of algorithms Heu, Appro, OL, DTP, CSP, Xanadu, Kraken and SC, by varying the memory capacity of each cloudlet from 6 GB to 14 GB while fixing the number of cloudlets at 200. The results are shown in Fig. 9 (a) and (b), from which we can see that the total costs and average delays of algorithms Heu, Appro, OL, DTP, CSP, Xanadu and Kraken decrease with the growth of the memory capacity of each cloudlet, while the total cost and the average delay of algorithm SC remain stable. The reason is that algorithm SC does not consider the memory capacity of each cloudlet. Specifically, when the memory capacity of each cloudlet is increased to 10 GB, Appro has the opportunity to execute all functions in the same cloudlet without violating its resource constraint, thus the total costs achieved by Appro and SC at this time are almost the same. Meanwhile, more functions can be placed within the proximity of users with the growth of the memory capacity of each cloudlet, which saving the data transmission costs and delays. It can be seen from Fig. 9 (c) that the weighted sum of cost and delay obtained by OL gradually decreases. Also, by allocating more memory resources to each cloudlet, each serverless function has a greater chance of being placed in a cloudlet with abundant memory without spending more time exploring the best location, thereby reducing the running time of algorithms, as shown in Fig. 9 (d).



(a) Total cost. (b) Average delay. (c) Weighted sum of cost and delay. (d) Running time.

Fig. 9. The impact of the memory capacity of each cloudlet on the performance of algorithms Heu, Appro, OL, DTP, CSP, Xanadu, Kraken and SC.



(a) Convergence. (b) Regret.

Fig. 10. The impact of the number of requests on the convergence and regret of algorithm OL.

We finally showed the convergence of algorithm OL as the number of requests increases from 1 to 100. From Fig. 10 (a), we can see that the weighted sum of cost and delay per request obtained by algorithm OL converge quickly as the number of requests increases. The reason is that the proposed algorithm OL can collect history data and learn in advance, resulting in better convergence. Specifically, by setting the values of ϵ to 0.1, 0.3, and 0.5, it can be seen that the algorithm OL converges when the number of requests is 20, 15, 10, respectively. Fig. 10 (b) shows the regret of algorithm OL, from which we can see that the regret is diminishing as the algorithm gradually converges. As such, our proposed algorithms can reduce the penalties, avoid higher costs, and reduce more delay violations even when the data volumes and network delays are uncertain.

VII. Conclusion and Future Work

In this paper, we studied a novel stateful serverless application placement problem in MEC networks. We first proposed an efficient heuristic for the problem. We then devised an approximation algorithm with provable approximation ratio for a special case of the problem without considering delay requirements. We also considered the dynamic stateful serverless application problem under the uncertainty of both data volume and network delays, and devised an online learning algorithm with a bounded regret. We finally evaluated the performance of the proposed algorithms by simulations. Simulation results show that the proposed algorithms outperform their counterparts by reducing at least 32% in the total cost and 27% in the average delay.

In the future work, we will explore the joint memory consolidation and function placement to reduce the size and number of memory fragments in cloudlets. Memory fragment incurred by the current methods is unavoidable. The placement of new functions and the consolidation of existing functions can both reduce the fragments. We thus will consider the design of efficient algorithms for jointly taking function placement and memory consolidation into consideration.

Acknowledgment

We appreciate the three anonymous referees and the associate editor for their constructive comments and valuable suggestions, which helped us improve the quality and presentation of the paper greatly.

References

[1] AWS Lambda Pricing. Accessed: Aug., 2022. [Online]. Available: https://aws.amazon.com/cn/lambda/pricing/
[2] AWS Step Functions. Accessed: Aug., 2022. [Online]. Available: https://aws.amazon.com/step-functions/
[3] N. Akhtar, A. Raza, V. Ishakian, and I. Matta, "COSE: Configuring serverless functions using statistical learning," in *Proc. IEEE Conf. Comput. Commun.*, 2020, pp. 129–138.
[4] O. Ascigil, A. Tasiopoulos, T. K. Phan, V. Sourlas, I. Psaras, and G. Pavlou, "Resource provisioning and allocation in function-as-a-service edge-clouds," *IEEE Trans. Serv. Comput.*, vol. 15, no. 4, pp. 2410–2424, Jul./Aug. 2022.
[5] I. Baldini et al., "Serverless computing: Current trends and open problems," *Research Advances in Cloud Computing*. Berlin, Germany: Springer, 2017, pp. 1–20.
[6] D. Barcelona-Pons et al., "On the FaaS Track: Building stateful distributed applications with serverless architectures," in *Proc. ACM 20th Int. Middleware Conf.*, 2019, pp. 41–54.
[7] L. Baresi and D. F. Mendonça, "Towards a serverless platform for edge computing," in *Proc. IEEE Int. Conf. Fog Comput.*, 2019, pp. 1–10.
[8] V. M. Bhasi et al., "Kraken: Adaptive container provisioning for deploying dynamic dags in serverless platforms," in *Proc. ACM Symp. Cloud Comput.*, 2021, pp. 153–167.
[9] P. Castro, V. Ishakian, V. Muthusamy, and A. Slominski, "The rise of serverless computing," *Commun. ACM*, vol. 62, no. 12, pp. 44–54, 2019.
[10] S. R. Chaudhry, A. Palade, A. Kazmi, and S. Clarke, "Improved QoS at the edge using serverless computing to deploy virtual network functions," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 10673–10683, Oct. 2020.
[11] L. Chen, C. Shen, P. Zhou, and J. Xu, "Collaborative service placement for edge computing in dense small cell networks," *IEEE Trans. Mobile Comput.*, vol. 20, no. 2, pp. 377–390, Feb. 2021.
[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. Cambridge, MA, USA: MIT Press, 2009.
[13] A. Das, S. Imai, S. Patterson, and M. P. Wittie, "Performance optimization for edge-cloud serverless platforms via dynamic task placement," in *Proc. IEEE/ACM 20th Int. Symp. Cluster Cloud Internet Comput.*, 2020, pp. 41–50.
[14] N. Daw, U. Bellur, and P. Kulkarni, "Xanadu: Mitigating cascading cold starts in serverless function chain deployments," in *Proc. ACM Int. Middleware Conf.*, 2020, pp. 356–370.
[15] T. Elgamal, A. Sandur, K. Nahrstedt, and G. Agha, "Costless: Optimizing cost of serverless computing through function fusion and placement," in *Proc. IEEE/ACM Symp. Edge Comput.*, 2018, pp. 300–312.
[16] GT-ITM. Accessed: Aug., 2022. [Online]. Available: http://www.cc.gatech.edu/projects/gtitm/

[17] B. Gao, Z. Zhou, F. Liu, and F. Xu, "Winning at the starting line: Joint network selection and service placement for mobile edge computing," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 1459–1467.
[18] T. He, H. Khamfroush, S. Wang, T. La Porta, and S. Stein, "It's hard to share: Joint service placement and request scheduling in edge clouds with sharable and non-sharable resources," in *Proc. IEEE 38th Int. Conf. Distrib. Comput. Syst.*, 2018, pp. 365–375.
[19] J. Jarachanthan, L. Chen, F. Xu, and B. Li, "Astra: Autonomous serverless analytics with cost-efficiency and QoS-awareness," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2021, pp. 756–765.
[20] S. G. Kolliopoulos and C. Stein, "Approximation algorithms for single-source unsplittable flow," *SIAM J. Comput.*, vol. 31, no. 3, pp. 919–946, 2001.
[21] Y. Liang, J. Ge, S. Zhang, J. Wu, Z. Tang, and B. Luo, "A utility-based optimization framework for edge service entity caching," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 11, pp. 2384–2395, Nov. 2019.
[22] Z. Li et al., "FaaSFlow: Enable efficient workflow execution for function-as-a-service," in *Proc. ACM Int. Conf. Archit. Support Program. Lang. Operating Syst.*, 2022, pp. 782–796.
[23] C. Lin and H. Khazaei, "Modeling and optimization of performance and cost of serverless applications," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 3, pp. 615–632, Mar. 2021.
[24] D. H. Lorenz and D. Raz, "A simple efficient approximation scheme for the restricted shortest path problem," *Operations Res. Lett.*, vol. 28, pp. 213–219, 2001.
[25] H. Ma, Z. Zhou, and X. Chen, "Leveraging the power of prediction: Predictive service placement for latency-sensitive mobile edge computing," *IEEE Trans. Wirel. Commun.*, vol. 19, no. 10, pp. 6454–6468, Oct. 2020.
[26] A. Mahgoub et al., "SONIC: Application-aware data passing for chained serverless applications," in *Proc. USENIX Annu. Tech. Conf.*, 2021, pp. 285–301.
[27] N. Mahmoudi, C. Lin, H. Khazaei, and M. Litoiu, "Optimizing serverless computing: Introducing an adaptive function placement algorithm," in *Proc. IEEE Annu. Int. Conf. Comput. Sci. Softw. Eng.*, 2019, pp. 203–213.
[28] G. McGrath and P. R. Brenner, "Serverless computing: Design, implementation, and performance," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. Workshops*, 2017, pp. 405–410.
[29] A. Palade et al., "A swarm-based approach for function placement in federated edges," in *Proc. IEEE Int. Conf. Serv. Comput.*, 2020, pp. 48–50.
[30] I. Pelle, F. Paolucci, B. Sonkoly, and F. Cugini, "Latency-sensitive edge/cloud serverless dynamic deployment over telemetry-based packet-optical network," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 9, pp. 2849–2863, Sep. 2021.
[31] K. Poularakis, J. Liorca, A. M. Tulino, I. Taylor, and L. Tassiulas, "Joint service placement and request routing in multi-cell mobile edge computing networks," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 10–18.
[32] M. Shahrad et al., "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider," in *Proc. USENIX Annu. Tech. Conf.*, 2020, pp. 205–218.
[33] J. Shen, H. Yu, Z. Zheng, C. Sun, M. Xu, and J. Wang, "Serpens: A high-performance serverless platform for NFV," in *Proc. IEEE/ACM 28th Int. Symp. Qual. Serv.*, 2020, pp. 1–10.
[34] The State Of Serverless. Accessed: Aug., 2022. [Online]. Available: https://www.datadoghq.com/state-of-serverless/
[35] M. Tiwary, P. Mishra, S. Jain, and D. Puthal, "Data aware web-assembly function placement," in *Proc. ACM Companion Web Conf.*, 2020, pp. 4–5.
[36] M. Wurster, U. Breitenbücher, K. Képes, F. Leymann, and V. Yussupov, "Modeling and automated deployment of serverless applications using TOSCA," in *Proc. IEEE 11th Conf. Service-Oriented Comput. Appl.*, 2018, pp. 73–80.
[37] Q. Xia, W. Liang, and W. Xu, "Throughput maximization for online request admissions in mobile cloudlets," in *Proc. 38th Annu. IEEE Conf. Local Comput. Netw.*, 2013, pp. 589–596.
[38] X. Xu et al., "Blockchain-based cloudlet management for multimedia workflow in mobile cloud computing," *Multimedia Tools Appl.*, vol. 79, pp. 9819–9844, 2019.
[39] F. Xu, Y. Qin, L. Chen, and F. Liu, "λDNN: Achieving predictable distributed DNN training with serverless architectures," *IEEE Trans. Comput.*, vol. 71, no. 2, pp. 450–463, Feb. 2022.
[40] Z. Xu, W. Liang, and Q. Xia, "Efficient embedding of virtual networks to distributed clouds via exploring periodic resource demands," *IEEE Trans. Cloud Comput.*, vol. 6, no. 3, pp. 694–707, Third Quarter 2018.
[41] Z. Xu, L. Zhou, Q. Xia, S. Chi-Kin Chau, W. Liang, and P. Zhou, "Collaborate of separate? Distributed service caching in mobile edge clouds," in *Proc. IEEE Conf. Comput. Commun.*, 2020, pp. 2066–2075.
[42] S. Yang, F. Li, S. Trajanovski, X. Chen, Y. Wang, and X. Fu, "Delay-aware virtual network function placement and routing in edge clouds," *IEEE Trans. Mobile Comput.*, vol. 20, no. 2, pp. 445–459, Feb. 2021.
[43] H. Zhang et al., "Fault-tolerant and transactional stateful serverless workflows," in *Proc. 14th USENIX Symp. Operating Syst. Des. Implementation*, 2020, pp. 1187–1204.
[44] L. Zhang et al., "Tapping into NFV environment for opportunistic serverless edge function deployment," *IEEE Trans. on Comput.*, vol. 71, no. 10, pp. 2698–2704, Oct. 2022, doi: 10.1109/TC.2021.3132776.
[45] Q. Zhang, Q. Zhu, M. F. Zhani, R. Boutaba, and J. L. Hellerstein, "Dynamic service placement in geographically distributed clouds," *IEEE J. Sel. Areas Commun.*, vol. 31, no. 12, pp. 762–772, Dec. 2013.

Zichuan Xu (Member, IEEE) received the BSc and ME degrees in computer science from the Dalian University of Technology in China in 2008 and 2011, and the PhD degree in computer science from the Australian National University in 2016. From 2016 to 2017, he was a research associate with the Department of Electronic and Electrical Engineering, University College London, U.K. He is currently a professor with the School of Software, Dalian University of Technology. He is also a "Xinghai Scholar' in Dalian University of Technology. His research interests include mobile edge computing, serverless computing, network function virtualization, Internet of Things, and algorithm design.

Lizhen Zhou received the BSc degree in computer science and technology from Tianjin University, China in 2019. She is currently working toward the PhD degree in Software Engineering with the Dalian University of Technology. Her research interests include mobile edge computing, Internet of Things, and network function virtualization.

Weifa Liang (Senior Member, IEEE) received the BSc degree in computer science from Wuhan University, China in 1984, the ME degree in computer science from the University of Science and Technology of China in 1989, and the PhD degree in computer science from the Australian National University in 1998. He is a professor with the Department of Computer Science, City University of Hong Kong. Prior to that, he was a professor with the Australian National University. His research interests include design and analysis of energy efficient routing protocols for Internet of Things, mobile edge computing (MEC), network function virtualization (NFV), software-defined networking (SDN), approximation algorithms, combinatorial optimization, and graph theory. He currently serves as an associate editor in the Editorial Board of *IEEE Transactions on Communications*.

Qiufen Xia (Member, IEEE) received the BSc and ME degrees in computer science from the Dalian University of Technology in China in 2009 and 2012, and the PhD degree in computer science from the Australian National University in 2017. She is currently an associate professor with the Dalian University of Technology. Her research interests include mobile cloud computing, query evaluation, Big Data analytics, Big Data management in distributed clouds, and cloud computing.

Wenzheng Xu (Member, IEEE) received the BSc, ME and PhD degrees in computer science from Sun Yat-Sen University, Guangzhou, PR China, in 2008, 2010, and 2015, respectively. He currently is an associate professor with the Sichuan University and was a visitor with the Australian National University. His research interests include wireless ad hoc and sensor networks, mobile computing, approximation algorithms, combinatorial optimization, online social networks, and graph theory.

Wenhao Ren received the BSc degree in computer science and technology from Harbin Engineering University in China in 2019. He is currently wirking toward the PhD degree with the Dalian University of Technology. His research interests include mobile cloud computing, Big Data processing, and network function virtualization.

Haozhe Ren received the BSc degree from the University of Science and Technology Beijing in China, in 2012, and the ME degree from the Xinjiang Normal University in China in 2018. He is currently working toward the PhD degree with the School of Software, Dalian University of Technology. His current research interests include network function virtualization, software-defined networking, algorithmic game theory, and optimization problems.

Pan Zhou (Senior Member, IEEE) received the BS degree in the advanced class of HUST, the MS degree from the Department of Electronics and Information Engineering, HUST, Wuhan, China, in 2006 and 2008, respectively, and the PhD degree from the School of Electrical and Computer Engineering, the Georgia Institute of Technology (Georgia Tech) in 2011, Atlanta, USA. He is currently a full professor and PhD advisor with Hubei Engineering Research Center on Big Data Security, School of Cyber Science and Engineering, Huazhong University of Science and Technology (HUST), Wuhan, P.R. China. He is currently an associate editor of *IEEE Transactions on Network Science and Engineering*. His current research interest includes: security and privacy, Big Data analytics, machine learning, and information networks.