Online Container Scheduling for Data-intensive Applications in Serverless Edge Computing

Xiaojun Shang*, Yingling Mao*, Yu Liu*, Yaodong Huang†, Zhenhua Liu‡, and Yuanyuan Yang*,
*Department of Electrical and Computer Engineering, Stony Brook University, USA
†College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China
‡Department of Applied Mathematics and Statistics, Stony Brook University, USA

IEEE INFOCOM 2023 - IEEE Conference on Computer Communications | 979-8-3503-3414-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/INFOCOM53939.2023.10229034

***Abstract*—Introducing the emerging serverless paradigm into edge computing could avoid over- and under-provisioning of limited edge resources and make complex edge resource management transparent to application developers, which largely facilitates the cost-effectiveness, portability, and short time-to-market of edge applications. However, the computation/data dispersion and device/network heterogeneity of edge environments prevent current serverless computing platforms from acclimating to the network edge. In this paper, we address such challenges by formulating a container placement and data flow routing problem, which fully considers the heterogeneity of edge networks and the overhead of operating serverless platforms on resource-limited edge servers. We design an online algorithm to solve the problem. We further show its local optimum for each arriving container and prove its theoretical guarantee to the optimal offline solution. We also conduct extensive simulations based on practical experiment results to show the advantages of the proposed algorithm over existing baselines.**

I. Introduction

The progressive penetration of Internet-of-Things (IoT) and the fast proliferation of artificial intelligence (AI) applications have endowed edge computing with unprecedented importance. Through deploying computing and storage capabilities near the network edge, many delay-sensitive IoT applications could be offloaded from the remote cloud to the proximity of edge users, which largely reduces service delay and backbone network pressure. In addition, edge computing enables a significant portion of data generated by IoT to be processed locally. Such a near-data paradigm greatly mitigates both transmission and privacy problems that impede the development of data-intensive services, e.g., AI applications.

Recently, serverless computing [1], [2], also known as Function-as-a-Service (FaaS) [3], introduces new inspirations to the edge computing landscape. In serverless computing, each application is realized as one or several functions that are initialized and executed upon a user event. It allows users to focus only on application development and offload all management operations, e.g., provisioning, scheduling, and scaling, to the service provider. It is believed by multiple researches, e.g., [4]–[7], that extending the serverless paradigm from the cloud to the network edge will bring new efficiency, flexibility, and scalability to tremendous edge-native applications.

The serverless service model applied in edge environments is often called serverless edge computing or deviceless computing [4], [5]. In such a paradigm, edge-native applications are implemented as functions and encapsulated in light-weighted containers, which could be flexibly started or stopped on edge servers according to dynamic workloads. This mechanism largely prevents resource over- or under-provisioning, which greatly alleviates resource shortage and enhances flexibility in edge environments. In addition, serverless edge computing makes the complex nature of edge resource management completely transparent to application developers. This feature significantly facilitates the portability and short time-to-market of edge applications, which are major challenges in today's edge computing realm.

To fuse the serverless service model into edge environments, systems and architectures have been proposed from both industry and academia, e.g., [6], [8]–[11]. Nevertheless, serverless edge computing platforms are still in their early stages, especially for data-intensive services such as edge AI applications, which are comprising the majority of serverless use cases [12], [13]. The major reason for such incompatibility comes from computation/data geo-dispersion and device/network heterogeneity of edge environments. Containers holding data-intensive applications often have close dependency on certain types and amounts of data during their initialization and execution. For instance, getting the corresponding image from a repository is necessary during the initialization of a container. Many containers holding functions such as training and inferring phases of AI applications also need to access large quantities of data during their execution periods. Unlike the cloud with centralized and homogeneous computation and storage substrates, supporting data-dependent containers on distributed and heterogeneous edge devices may incur large transmission and execution delays that offset the benefits of the serverless paradigm. On one hand, placing containers near data origins may lead to significant execution delay, since edge devices often have limited resources and may not have desired accelerators to speed up computation. On the other hand, transmitting data to edge devices or the remote cloud with sufficient computing resources may lead to large transmission delays and privacy issues.

There exists pioneering work proposing data-aware container orchestration strategies and systems, e.g., [11], to bridge the gaps between the serverless paradigm and the network edge for data-intensive applications. Nonetheless, existing solutions are not sufficient to address the aforementioned challenges, since multiple unique features of the geo-distributed and heterogeneous serverless edge computing network have not been fully considered. First, as far as we are concerned, existing work has not considered the heterogeneous network

typologies at the network edge that may be much different from the cloud. For example, an edge server holding containers and the corresponding data location may not be directly connected by one router but a complex routing path. Thus, besides container scheduling, how to jointly route data flows between containers and data sources for low transmission delay is complex but pivotal towards a successful container orchestration system. In addition, despite the efforts of simplifying serverless computing platforms for edge environments such as the K3s build [14], extra energy and resources for container orchestration and maintenance are still needed, e.g., using a container scheduler to deploy containers and running daemons like Kubelet [15] to take care of detailed starting, stopping, and maintaining containers on each edge server. Considering both energy and resource are highly limited at the network edge, such extra operating costs for serverless edge computing should not be omitted. Furthermore, since serverless operating costs exist, initializing a container in the remote cloud and transmitting corresponding data to it is not necessarily inferior to starting a container near the data location at the edge. It is highly desirable to have a model that trades off delays and operating costs over the edge and the cloud to achieve optimal container scheduling results. Finally, existing container orchestration systems schedule containers using greedy algorithms with no performance guarantee in the worst cases. It is thus desirable to come up with an online algorithm balancing both performance and theoretical guarantee.

In this paper, we design an online container scheduling strategy for data-intensive applications to solve the problems in serverless edge computing mentioned above. Our main contributions are summarized as follows.

- To overcome the computation/data dispersion and device/network heterogeneity in serverless edge computing, we formulate a joint container placement and flow routing problem. The formulated model reflects diverse execution and transmission delays of data-intensive applications on different edge devices. It also takes various network typologies at the edge into consideration for detailed data flow routing. Meanwhile, the formulation makes it possible to trade off the operating costs of enabling the serverless edge computing paradigm and the delay reduction gained by deploying containers at the network edge. In this way, an optimized container scheduling over the edge and the cloud could be achieved.
- To solve the formulated problem, we design an Online Data-aware Container Scheduling (ODCS) algorithm that jointly places containers and routes data flows when container requests arrive in a sequential manner. We then show in the theoretical analysis that the placement and routing result is locally optimized for each arriving container. We further prove a theoretical bound of the ODCS algorithm to the optimal offline solution in the worst cases.
- We verify the advantages of our algorithm through extensive simulations, the settings of which are based on data from small-scale practical experiments. Simulation results show that the proposed ODCS algorithm achieves much better performance than existing baselines.

The remainder of this paper is organized as follows. Section II presents an overview of related work. Section III proposes the joint container placement and flow routing problem while Section IV demonstrates the ODCS algorithm and corresponding theoretical analysis. Section V further shows the results of the small-scale experiment and extensive simulations. Finally, Section VI concludes the paper.

II. Related Work

Lately, serverless computing is drawing more and more attention from both industry and academia because of its pay-as-you-go pricing model, low complexity, cost-effectiveness, auto-scaling characteristics, etc. Several tech giants have developed commercial products such as Azure Functions [16], Cloud Functions [17], Lambda [18], and OpenWhisk [19]. Many research studies have also been proposed focusing on current challenges and open issues of serverless computing, e.g., [1], [20]–[24]. Among them, one significant topic is how to adapt the serverless computing paradigm to edge environments given its great potential in numerous application scenarios, e.g., smart home, edge swarms, industry 4.0, and urban sensing [11], [25].

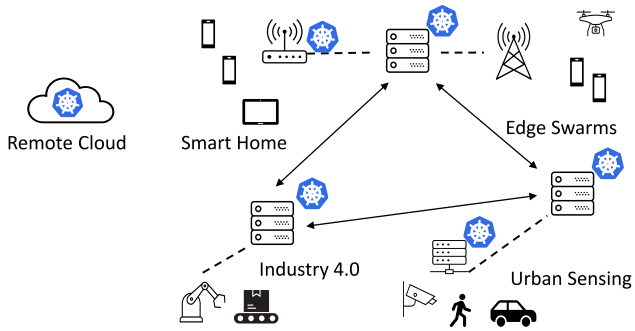


Fig. 1. Demonstration of a serverless edge computing network. Here, we utilize the logo of Kubernetes (a blue ship wheel) to represent that a server supports serverless computing.

In industry, big tech companies have already set their sights on serverless edge computing. For instance, Amazon proposed AWS IoT Greengrass [26] that could carry out AWS Lambda functions on edge devices, aiming at bringing intelligence to the network edge. In academia, algorithmic, systematic, and architectural researches have also been proposed. Glikson et al. pointed out major requirements of serverless edge computing and defined a deviceless paradigm to emphasize the differences between serverless models at the edge and in the cloud [5]. Aslanpour et al. put forward a detailed analysis concerning the opportunities, e.g., always-on mitigation, event-driven applications, pure pay per use, and open issues, e.g., cold starts, distributed networking, edge artificial intelligence, in [13]. Xiong et al. extended a series of components in Kubernetes to adapt

the container orchestration platform to edge environments and named the modified system KubeEdge [8]. While Nastic et al. proposed a serverless real-time data analytics platform in [27], Baresi et al. constructed a practical network architecture for serverless edge computing in [6]. According to these pioneering researches, a serverless edge computing platform could be concluded as an ingenious combination and extension of multiple leading-edge virtualization technologies, e.g., docker [28], OpenFaaS [29], and Kubernetes [30], which is deployed on interconnected edge devices to provide low-delay, highly flexible, and cost-effective edge-native applications as shown in Fig 1.

To mitigate the negative impacts of computation/data geo-distribution to a serverless edge computing network as mentioned in Section I, Pan et al. formulated in [31] a container caching jointly with request distribution problem and designed online algorithms to solve it. Rausch et al. proposed a data-aware container orchestration system in [11] to acclimate data-intensive edge functions to edge environments. Nevertheless, existing work omits the necessity of data flow routing given various and heterogeneous edge network typologies and does not take the operating costs of serverless edge platforms into consideration. Mechanisms focusing on end-to-end service allocation and flow routing, e.g., [32]–[35], on the other hand, could not be directly applied in the new serverless edge computing scenario due to constraints such as data availability and privacy requirements. Therefore, we formulate the joint container placement and data routing problem for serverless edge computing in this paper and provide an online solution with a theoretical guarantee to the optimal offline solution.

III. Model Formulation

In this section, we formulate the joint container placement and flow routing problem. We consider a serverless edge computing network for data-intensive applications shown as Fig. 2. For unification, we call all units in the network that could hold containers "nodes" and denote the set of nodes as $N = \{1, ..., n, ..., |N|\}$. It is worth noting that we use the node 1 to represent the remote cloud and other nodes as edge servers. We also use $I = \{1, ..., i, ..., |I|\}$ to denote the set of containers to be scheduled on these nodes. To formulate a container scheduling problem that targets mitigating the negative impacts of computation/data geo-distribution and device/network heterogeneity to serverless edge computing, we need to consider in total four sets of decision variables in the model.

The variable $x_{i,n} \in \{0, 1\}$ in Set X represents whether the i^{th} container is deployed on node n. The variable $y_{i,n} \in \{0, 1\}$ in Set Y represents whether the data for the i^{th} container is retrieved from node n. Due to the heterogeneous topology of edge networks, we define $e_{n_1,n_2} \in E$ as the communication link between nodes n_1 and n_2 and utilize the variable $f_{i,n_1,n_2} \in \{0, 1\}$ to represent if the flow between any container i and its corresponding data passes through link e_{n_1,n_2} or not. In addition, since realizing the serverless service model on an edge node involves extra operating costs,

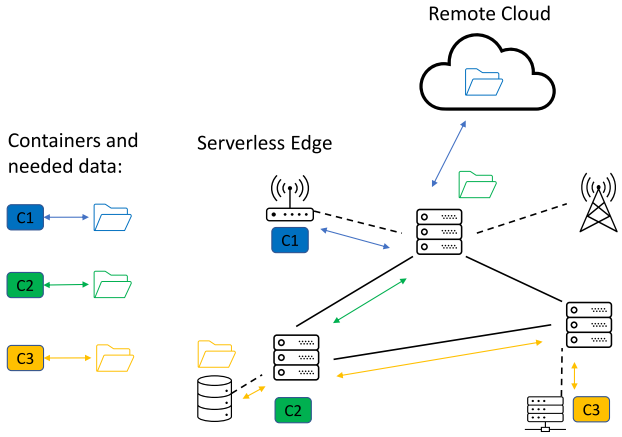


Fig. 2. Demonstration of the container placement and data flow routing problem for data-intensive applications in a serverless edge computing network. Different containers are represented by C_1, C_2, and C_3, while their corresponding data are file symbols with corresponding colors. To solve the problem, the container scheduler needs to determine the placement of each container and decide which data location to choose for data retrieve and which flow routing path to follow.

as mentioned in Section I, we consider $z_n = \{0, 1\}$ as the variable indicating whether the node n is enabled to support the serverless platform. With the decision variables defined, we now construct the objective function with multiple terms as follows. All important notations are listed in Table 1.

TABLE I

Notation	Definition
N	Set of nodes in the network, $N = \{1, ..., n, ..., \|N\|\}$.
I	Set of containers in the network, $I = \{1, ..., i, ..., \|I\|\}$.
e_{n_1,n_2}	The communication link connecting node n_1 to n_2.
E	Set of communication links.
$x_{i,n} \in \{0, 1\}$	Decision variable whether container i is placed on node n.
$y_{i,n} \in \{0, 1\}$	Decision variable whether data of container i is retrieved from node n.
$z_n \in \{0, 1\}$	Decision variable whether node n is enabled for serverless computing.
$f_{i,n_1,n_2} \in \{0, 1\}$	Decision variable whether flow between container i and corresponding data passes through e_{n_1,n_2}.
w_{i,n_1,n_2}	The transmission delay introduced by the flow between container i and corresponding data passing through e_{n_1,n_2}.
$\alpha_{i,n}$	Execution delay introduced by placing container i on node n.
$\beta_{i,n}$	Constant marking the availability of data for container i on node n.
γ_n	Operating cost of node n enabling serverless computing.
p	Cost of operating a container with one unit of resource consumption in the cloud.
c_i	Resource consumption of container i.
C_n	Resource capacity of node n.

Transmission delay: Transmission delay occurs when data is transmitted from one node to another. We thus define a parameter w_{i,n_1,n_2} representing the transmission delay introduced by the data flow between the container i and the chosen data source passing the communication link e_{n_1,n_2}. With each flow variable f_{i,n_1,n_2} defined, the total data transmission delay introduced by all containers could be summarized as

$$U_1 = \sum_{i \in I} \sum_{n_1 \in N} \sum_{n_2 \in N} w_{i,n_1,n_2} \cdot f_{i,n_1,n_2}.$$

Execution delay: Due to the heterogeneity of edge computing, different edge devices have various types and amounts of resources. Hence, it is possible that the same container will incur distinct execution delays on different nodes. According to our experiment results shown in Section V, a container encapsulated with a machine learning inference function could have about 3.6 times execution delay on a Raspberry Pi 4 device compared to the same container on a Jetson Xavier NX device with a GPU. We thus define the execution delay of container i on node n as $\alpha_{i,n}$. We could then formulate the total execution delay as

$$U_2 = \sum_{i \in I} \sum_{n \in N} \alpha_{i,n} \cdot x_{i,n}.$$

It is worth noting that $\alpha_{i,n}$ could be set to infinity to represent that the data of container i could not be transmitted to node n due to constraints such as privacy issues. Hence, our model is compatible with applications that have privacy or security requirements.

Data availability: For the availability of needed data for each container, we consider the most generalized case in this model. First, data could be located at edge nodes. It could be a container image that is downloaded and saved in the local storage of an edge node. It could also be raw data generated by sensors that is collected and stored by an edge storage server nearby. In addition, data could also be retrieved from the remote cloud that contains numerous container repositories and databases. Moreover, data needed by a container i may exist on multiple nodes at the same time. In this way, we use a parameter $\beta_{i,n}$ to represent whether the data needed by container i could be retrieved from node n. If so, $\beta_{i,n} = 0$. Otherwise, $\beta_{i,n}$ will be assigned infinity to mark its unavailability. In this way, we formulate the data availability for all containers in the objective function as

$$U_3 = \sum_{n \in N} \sum_{i \in I} \beta_{i,n} \cdot y_{i,n}.$$

Operating cost: As mentioned in Section I, applying the serverless service model involves operating costs from multiple aspects such as container scheduling and daemons. Since both energy and computation resources are highly limited at the network edge, such operating costs often have worse impacts at the network edge than that in the cloud, which means larger relative values in the model. For an edge node n, we define γ_n as the operating cost if the node is enabled to hold containers for serverless edge computing. γ_n varies for different types of nodes due to the device heterogeneity. Besides, we consider the operating cost of a container placed in the cloud is proportional to its resource usage c_i and utilize the parameter p to represent the positive ratio. Therefore, the total operating cost of the serverless edge computing network could be considered as

$$U_4 = \sum_{n \in N/1} \gamma_n \cdot z_n + \sum_{i \in I} p \cdot c_i \cdot x_{i,1}.$$

With the objective function determined, we further consider the constraints for the container placement and data flow routing problem. First, each edge node has a maximal resource capacity for containers. Denote by c_i and C_n the resource needed by container i and the resource capacity of node n. It is obvious that any node could hold containers only if it is enabled for serverless computing, which is indicated by variable z_n. We thus have resource constraints defined as

$$\sum_{i \in I} c_i \cdot x_{i,n} \leq z_n \cdot C_n, \forall n \in N. \tag{1}$$

Here, C_1 is a very large positive constant ($\geq \sum_{i \in I} c_i$) so that cloud can supply enough resource for all containers. It is also clear that each container will only be placed once on a single node, we thus have

$$\sum_{n \in N} x_{i,n} = 1, \forall i \in I. \tag{2}$$

Similarly, although data for a certain container may be available at multiple locations, it only need to be retrieved from one location, so that

$$\sum_{n \in N} y_{i,n} = 1, \forall i \in I. \tag{3}$$

In addition, the flows among containers and corresponding data should satisfy the flow conservation law on any type of network topology. This means that, for each data flow, the in-degree and out-degree of a node n should be the same if both container i and its data are on the node or neither of them is on the node. Otherwise, there will be one unit of difference. Hence, we can conclude that

$$\sum_{n_1 \in N} f_{i,n_1,n} - \sum_{n_2 \in N} f_{i,n,n_2} = x_{i,n} - y_{i,n}, \forall n \in N. \tag{4}$$

In the end, we also need to ensure that all variables are binary in the formulated model that

$$x_{i,n}, y_{i,n}, z_n, f_{i,n_1,n_2} \in \{0,1\}, \forall i \in I, n, n_1, n_2 \in N. \tag{5}$$

With the objective function and constraints defined, we hence formulate the data-aware container placement and flow routing problem $P1$ as follows.

$$\begin{aligned} \min_{x_{i,n}, y_{i,n}, z_n, f_{i,n_1,n_2}} \quad & U_1 + U_2 + U_3 + U_4 \\ \text{s.t.} \quad & (1),\ (2),\ (3),\ (4),\ (5). \end{aligned} \tag{P1}$$

It is worth noting that the unit of U_4 is different from the first two terms in the objective function of $P1$. U_3 is a penalty item, which is either zero or infinity. Related work has proposed methods to solve such a problem for multi-objective problems. For instance, Rausch et al. designed a dedicated simulator to find the best relative value of each objective term in [11]. We will not consider the details in this paper due to the page

limitation. Nonetheless, we will prove in the following section IV that the algorithm we propose in this paper will always have a theoretical bound to the optimal offline solution. We will also show that the superiority of our algorithm over other baselines preserves when the relative value of operating cost changes through extensive simulations in Section V.

IV. Online Data-aware Container Placement and Flow Routing

In this section, we will discuss the hardness of the formulated problem $P1$. We will then demonstrate an online algorithm specifically designed to solve $P1$ with low complexity. In the end, we will show that the proposed algorithm is locally optimal for each arriving container and has a theoretical bound to the optimal offline solution.

A. The Hardness of P1

By assigning parameters w_{i,n_1,n_2}, $\alpha_{i,n}$, $\beta_{i,n}$, and p to zero, and γ_n to one, we can reduce the well-known bin packing problem to the problem $P1$. Therefore, $P1$ is at least strongly NP-complete even if container requests come in a batch and the problem could be solved in an offline manner. Furthermore, since applications in serverless computing are realized as functions that are initialized and executed upon users' events, containers holding such functions are often started in an online manner. We thus need to consider a more complex case for $P1$ that the container requests will arrive sequentially instead of all at once. This means that corresponding information about the container i is unknown until its arrival. Such missing information makes it even harder to solve $P1$ with close performance to the optimal offline solution, which knows all future information in advance.

B. Online Algorithm Design

In view of the hardness and online features of $P1$, we design the Online Data-aware Container Scheduling (ODCS) algorithm to handle the placement and data flow routing of each arriving container request i. The ODCS algorithm considers the serverless edge computing network as a directed graph $G_i = (N, E)$, where the edge weight of each communication link e_{n_1,n_2} from node n_1 to n_2 is w_{i,n_1,n_2}. Suppose that all nodes with necessary data for container i belong to a node set N_i^{data}. Here, $\beta_{i,\hat{n}_k} = 0$ if $\hat{n}_k \in N_i^{data}$. For each $\hat{n}_k$, ODCS runs a single-sourced shortest path algorithm and saves the shortest path and the corresponding path length $l_{i,\hat{n}_k,n}$ from $\hat{n}_k$ to every node n in G_i. The computational complexity of such a procedure for each $\hat{n}_k$ is $\mathcal{O}((|N| + |E|)\log(|N|))$. The same process is carried out for all possible data locations $\hat{n}_k \in N_i^{data}$ with the total complexity at most $\mathcal{O}((|N|+|E|)|N|\log(|N|))$ in the case that data needed by container i is available on every node in the network. Then, for each possible container location n, the data location $\hat{n}_k$ with the smallest $l_{i,\hat{n}_k,n}$ is chosen to retrieve data for container i along the corresponding routing path. Denote by $h_{i,n}$ the transmission delay along the chosen routing path from data to container i if it is placed on node n, i.e., the smallest path length $l_{i,\hat{n}_k,n}$ for any $\hat{n}_k$, we can transform the original $P1$ into a new online problem $P2$ by combining $h_{i,n}$ and $\alpha_{i,n}$ and eliminating $U1$ and $U3$ as follows.

$$\min_{x_{i,n},z_i} \quad \sum_{i\in I}\sum_{n\in N}\alpha'_{i,n}\cdot x_{i,n} + \sum_{n\in N/1}\gamma_n\cdot z_n + \sum_{i\in I} p\cdot c_i\cdot x_{i,1}$$
$$\text{s.t.} \quad (1),\ (2),\ (5). \tag{P2}$$

Here, $\alpha'_{i,n} = \alpha_{i,n} + h_{i,n}$. Although simplified, $P2$ is still at least strongly NP-complete even with batched arrivals according to a similar proof. To solve $P2$ for each arriving container i, the ODCS algorithm in total has three types of choices. The first choice is to place the container on an edge node that has been initialized for serverless computing and has sufficient remaining resource capacity. The second choice is to initialize a new edge node and deploy the container on it. The third choice is to start the container in the remote cloud. In our design, the metric for the ODCS algorithm to make the decision is the increment to the objective function δ_n by placing container i on node n. In the first type of choice, the increment to the objective function $\delta_{i,n} = \alpha'_{i,n}$. For the second choice, $\delta_{i,n} = \alpha'_{i,n} + \gamma_n$. If the container is to be placed in the cloud in the third choice, $\delta_{i,1} = \alpha'_{i,1} + c_i \cdot p$. The ODCS sorts all nodes in the increasing order of $\delta_{i,n}$ and deploys container i into the first fit node with sufficient capacity. In general, the complexity of running the ODCS algorithm for each container is $\mathcal{O}((|N|+|E|)|N|\log(|N|))$ and the total complexity for $|I|$ containers is thus $\mathcal{O}((|N|+|E|)|N||I|\log(|N|))$. More details about the online algorithm could be found in Algorithm 1.

C. Theoretical Analysis

We now analyze the proposed online algorithm theoretically and show that it is local optimal for each arriving container request.

Theorem 1. *For each request i, the container placement and routing decision of the ODCS algorithm is local optimum.*

Proof. It is clear that by running the shortest path algorithm and choosing the data location with the lowest transmission delay $h_{i,n}$ for each possible container placement $x_{i,n}$ (lines 4-9 in Algorithm 1), choosing the optimal data location and routing path is converted to choosing the best $x_{i,n}$ to minimize $h_{i,n} \cdot x_{i,n}$. Therefore, transforming $P1$ to $P2$ by combining $h_{i,n} \cdot x_{i,n}$ and $\alpha_{i,n} \cdot x_{i,n}$ will not change the local optimal direction of the online problem. Since the problem $P2$ is then solved by the ODCS algorithm via choosing the container placement with the lowest objective value, the result is locally optimized for each arriving container request i. □

We then prove that the global performance of the ODCS algorithm has a theoretical guarantee to the optimal offline solution.

Theorem 2. *Suppose the solution of the ODCS algorithm is $S^\dagger$ and we have S^* as the result of the optimal offline solution. Then, the competitive ratio between ODCS and the offline*

Algorithm 1 The ODCS Algorithm

Input: Related information of the upcoming containers, i.e., $\alpha_{i,n}$, $\beta_{i,n}$, w_{i,n_1,n_2}, c_i. The conditions of the edge-cloud network, i.e., C_n, γ_n, p.
Output: Container placement $x_{i,n}$, data location choice $y_{i,n}$, flow routing f_{i,n_1,n_2}, and enabled nodes for serverless computing z_n.

for all $i \in I$ **do**
  Construct a directed graph G_i with edge weight w_{i,n_1,n_2}.
  Suppose all nodes $\hat{n}_k$ with $\beta_{i,\hat{n}_k} = 0$ belong to Set N_i^{data}.
  for all $\hat{n}_k \in N_i^{data}$ **do**
    Conduct the single-sourced shortest path algorithm for node $\hat{n}_k$ and get the shortest path and corresponding length $l_{i,\hat{n}_k,n}$ to each node n in G_i.
  end for
  for all $n \in N$ **do**
    Find the $\hat{n}_k \in N_i^{data}$ with the smallest $l_{i,\hat{n}_k,n}$ and assign $h_{i,n} = \min_{\hat{n}_k}\{l_{i,\hat{n}_k,n}\}$.
    Substitute $\alpha_{i,n}$ with $\alpha'_{i,n} = \alpha_{i,n} + h_{i,n}$ and eliminate U_1 and U_3 to formulate $P2$.
  end for
  Suppose the increment to the objective function of $P2$ by placing container i on node n is $\delta_{i,n}$.
  for all $n \in N$ **do**
    if $n = 1$ **then**
      $\delta_{i,1} = \alpha'_{i,1} + c_i \cdot p$
    else if $z_n = 0$ **then**
      $\delta_{i,n} = \alpha'_{i,n} + \gamma_n$
    else
      $\delta_{i,n} = \alpha'_{i,n}$
    end if
    Sort nodes in the increasing order of $\delta_{i,n}$ as Set N_i^{sort}.
    Denote the current workload of node n is L_n.
    for all $n \in N_i^{sort}$ **do**
      if $L_n + c_i \leq C_n$ **then**
        $x_{i,n} = 1$, $L_n = L_n + c_i$.
        $y_{i,\hat{n}_{k'}} = 1$, where $l_{i,\hat{n}_{k'},n} = h_{i,n}$.
        if $z_n = 0$ **then**
          $z_n = 1$.
        end if
        Assign f_{i,n_1,n_2} to 1 along the shortest path from $\hat{n}_{k'}$ to n in G_i.
        Break.
      end if
    end for
  end for
end for

optimum is a constant, i.e., $S^\dagger \leq \eta \cdot S^*$. Here, the constant η will be defined in the following proof.

Proof. For each node n, we consider three sets of containers $I_n^\dagger$, I_n^*, and I_n^{first} placed on it. Set $I_n^\dagger$ and I_n^* contain the containers that are placed on node n by the ODCS algorithm and the optimal offline solution, respectively. Set I_n^{first} includes all containers that would be placed on node n if there is no capacity constraint on the node n. To prove the theorem, we need to consider three different cases for each node n.

Case 1, when $n = 1$: This means the node is the remote cloud. We define the portion of objective value contributed by containers in case 1 by the OCPS algorithm as $alg_{1,1}$ that

$$alg_{1,1} = \sum_{i \in I_1^\dagger} (\alpha'_{i,1} + c_i \cdot p) = \sum_{i \in I_1^\dagger} \frac{\alpha'_{i,1}}{c_i}(1 + \frac{c_i}{\alpha'_{i,1}} \cdot p) \cdot c_i.$$

Consider $\theta_{max} = \max(\frac{\alpha'_{i,n}}{c_i})$ and $\theta_{min} = \min(\frac{\alpha'_{i,n}}{c_i})$, it is clear that

$$alg_{1,1} \leq \theta_{max}(1 + \frac{p}{\theta_{min}}) \cdot \sum_{i \in I_1^\dagger} c_i.$$

Similarly, we have $opt_{1,1}$, the objective value of the optimal offline solution

$$opt_{1,1} \geq \theta_{min}(1 + \frac{p}{\theta_{max}}) \cdot \sum_{i \in I_1^*} c_i.$$

Case 2, when $I_n^{first} \subseteq I_n^\dagger$ and $n \neq 1$: This means the node n has sufficient capacity to include all containers which will contribute to the minimized objective value if placed on the node. Then we have the objective value of our algorithm for each n, $alg_{2,n}$, as

$$alg_{2,n} = \sum_{i \in I_n^\dagger} \alpha'_{i,n} + \gamma_n = \sum_{i \in I_n^{first}} \alpha'_{i,n} + \gamma_n + \sum_{i \in I_n^\dagger / I_n^{first}} \alpha'_{i,n}.$$

Then, it is clear that

$$alg_{2,n} \leq \sum_{i \in I_n^{first}} \alpha'_{i,n} + \gamma_n + \theta_{max} \cdot \sum_{i \in I_n^\dagger / I_n^{first}} c_i.$$

For any node n in case 2, since its capacity is sufficient to include all containers in I_n^{first}, the optimal offline solution will definitely also have $I_n^{first} \subseteq I_n^*$. Therefore, we have a similar conclusion that

$$opt_{2,n} \geq \sum_{i \in I_n^{first}} \alpha'_{i,n} + \gamma_n + \theta_{min} \cdot \sum_{i \in I_n^* / I_n^{first}} c_i.$$

Case 3, when $I_n^{first} \nsubseteq I_n^\dagger$ and $n \neq 1$: In this case, we have the objective value of our algorithm for each n, $alg_{3,n}$, as

$$alg_{3,n} = \sum_{i \in I_n^\dagger} \alpha'_{i,n} + \gamma_n = \left(1 + \frac{C_n}{\sum_{i \in I_n^\dagger} \alpha'_{i,n}} \cdot \frac{\gamma_n}{C_n}\right) \cdot \sum_{i \in I_n^\dagger} \alpha'_{i,n}.$$

Consider $\sigma_{max} = max(\frac{\gamma_n}{C_n})$ and $\sigma_{min} = min(\frac{\gamma_n}{C_n})$, we have

$$alg_{3,n} \leq \left(1 + \frac{C_n}{\sum_{i \in I_n^\dagger} \alpha'_{i,n}} \cdot \sigma_{max}\right) \cdot \sum_{i \in I_n^\dagger} \alpha'_{i,n}$$

$$\leq \left(1 + \frac{C_n}{\theta_{min} \cdot \sum_{i \in I_n^\dagger} c_i} \cdot \sigma_{max}\right) \cdot \sum_{i \in I_n^\dagger} \alpha'_{i,n}.$$

Based on the observation of our practical experiment in Section V, we assume that the node with the smallest capacity can still hold the container with the maximal resource consumption, i.e., $R = \frac{\min(C_n)}{\max(c_i)} \geq 1$. Since $I_n^{first} \nsubseteq I_n^\dagger$, there exists at least one container $\hat{i}$ that $\hat{i} \in I_n^{first}$ and $\hat{i} \notin I_n^\dagger$ due to the capacity limitation. Hence, we have

$$\sum_{i \in I_n^\dagger} c_i + c_{\hat{i}} > C_n.$$

According to the definition of R, we know that

$$c_{\hat{i}} \leq \max(c_i) = \frac{1}{R} \cdot \min(C_n) \leq \frac{C_n}{R}.$$

Then, it is clear that

$$\sum_{i \in I_n^\dagger} c_i > C_n - c_{\hat{i}} \geq C_n - \frac{C_n}{R} = \frac{R-1}{R} C_n.$$

Thus, we have

$$alg_{3,n} \leq \left(1 + \frac{C_n}{\theta_{min} \cdot \frac{R-1}{R} C_n} \cdot \sigma_{max}\right) \cdot \sum_{i \in I_n^\dagger} \alpha'_{i,n}$$

$$\leq \theta_{max} \cdot \left(1 + \frac{R}{R-1} \cdot \frac{\sigma_{max}}{\theta_{min}}\right) \cdot \sum_{i \in I_n^\dagger} c_i.$$

Substituting $\sum_{i \in I_n^\dagger} c_i > \frac{R-1}{R} C_n$ with $\sum_{i \in I_n^*} c_i < C_n$, we can also conclude that

$$opt_{3,n} \geq \theta_{min} \cdot \left(1 + \frac{\sigma_{min}}{\theta_{max}}\right) \cdot \sum_{i \in I_n^*} c_i.$$

Taking all three cases into consideration, we can summarize the competitive ratio between our algorithm and the optimal offline solution as

$$\frac{S^\dagger}{S^*} = \frac{\sum_{n \in case1} alg_{1,1} + \sum_{n \in case2} alg_{2,n} + \sum_{n \in case3} alg_{3,n}}{\sum_{n \in case1} opt_{1,1} + \sum_{n \in case2} opt_{2,n} + \sum_{n \in case3} opt_{3,n}}$$

It is worth noting that $\sum_{n \in case2} alg_{2,n}$ and $\sum_{n \in case2} opt_{2,n}$ share the same component $m = \sum_{n \in case2} \sum_{i \in I_n^{first}} \alpha'_{i,n} + \gamma_n \geq 0$. Knowing that if $A \geq B > m \geq 0$, then $\frac{A}{B} \leq \frac{A-m}{B-m}$. We hence can conclude that $\frac{S^\dagger}{S^*}$ is less than or equal to

$$\frac{\sum_{n \in case1} alg_{1,1} + \sum_{n \in case2} alg_{2,n} + \sum_{n \in case3} alg_{3,n} - m}{\sum_{n \in case1} opt_{1,1} + \sum_{n \in case2} opt_{2,n} + \sum_{n \in case3} opt_{3,n} - m}.$$

Here, we consider all containers counted in m belong to the set $\hat{I}$. Therefore,

$$\frac{S^\dagger}{S^*} \leq \frac{\theta_{max} \cdot (1 + \frac{\max\{p, 0, \frac{R \cdot \sigma_{max}}{R-1}\}}{\theta_{min}}) \cdot \sum_n \sum_{i \notin \hat{I}} c_i}{\theta_{min} \cdot (1 + \frac{\min\{p, 0, \sigma_{min}\}}{\theta_{max}}) \cdot \sum_n \sum_{i \notin \hat{I}} c_i}$$

$$= \frac{\theta_{max}}{\theta_{min}} \cdot (1 + \frac{\max\{p, \frac{R \cdot \sigma_{max}}{R-1}\}}{\theta_{min}}) = \eta.$$

□

After proving the theoretical guarantee of the ODCS algorithm to the optimal offline solution in the worst cases, we then evaluate its average performance compared to existing baselines in the following section.

V. Performance Evaluation

To highlight the advantages of the proposed ODCS algorithm, we conduct extensive simulations to compare it with existing container scheduling algorithms for serverless edge computing. Many basic settings and data in the simulations are based on small-scale experiments, in which several real-world data-intensive applications are implemented as serverless functions on representative edge devices. Data transmission is also carried out among different edge devices in the same edge network. In this section, we will first briefly illustrate the small-scale experiments and then demonstrate the simulation settings based on them. Finally, we will present and discuss the simulation results.

A. Serverless Edge Computing Experiments

The experiments are realized in an edge network consisting of two types of edge nodes, raspberry Pi 4 [36] and Jetson Xavier NX [37]. There are in total 4 Pi 4 and 2 NX devices and all devices are connected in an Ad-Hoc network. Each raspberry Pi 4 has 8G memory and 32G storage, while each Jetson Xavier NX has 16G memory and 128G storage. We utilize a lightweight container orchestration system, i.e., K3s [14], to schedule and manage containers among the edge devices. K3s is a highly available, certified Kubernetes distribution designed for production workloads in unattended, resource-constrained, remote locations or inside IoT appliances. We implement three types of data-intensive serverless functions on the serverless edge computing framework, i.e., data pre-processing, machine learning inference with a convolutional neural network (CNN), and results feedback to end users. Such three serverless functions all involve data transmission with non-negligible delays in the edge network. During the experiments, we record the average execution and data transmission delays for each serverless function as shown in Table II.

TABLE II

	Function 1	Function 2	Function 3
Execution delay (s)	0.0027	1.1432	0.0001
Transmission delay (s)	1.4950	1.5926	0.2985

Besides above values, we also abstract heterogeneous features of serverless edge computing from the experiments for our simulation. For instance, the average execution delay of the machine learning inference function on a raspberry pi 4 is 1.7520 seconds. However, the container running the same function on a Jetson Xavier NX device only takes 0.4883 second to execute. More details about how to convert such experiment results to simulation settings will be discussed in the following section.

B. Simulation Settings

Due to the fact that edge networks may have different topologies, we apply our ODCS algorithm on randomly generated connected graphs in the simulations. The connectivity is 0.2 by default but varies in different simulations to represent the network heterogeneity. In the experiments, we observe that the capacity of each edge node is mainly constrained by its memory resource when executing data-intensive applications. Given the memory resource of our edge devices is 8G and 16G, we set the capacity of each edge node C_n randomly distributed in range $[6, 14]$. Here, 2G memory is set aside for system functions. According to the minimal and maximal memory consumption of different containers in our experiments, i.e., 200M and 1.5G, we set the range of the resource requirement of each container c_i as $[0.2, 1.5]$. We further set the execution delays, i.e., $\alpha_{i,n}$, as the benchmark of the objective function. The basic value of this parameter for each container i is uniformly distributed between $[0.0001, 1.1432]$ based on Table II. While for different edge node n, we multiply the basic value with a factor that is randomly distributed in $[1, \frac{1.7520}{0.4883}]$, which represents the execution delay difference

between Raspberry Pi 4 and Jetson Xavier NX, to imitate the delay variety on different edge devices.

For the transmission delays, we consider the delays among edge nodes are uniformly distributed in range $[0.2985, 1.5926]$. Since we do not include the transmission delay from the edge to the cloud in the current experiments, we apply a ratio of edge-to-cloud delays to edge-to-edge delays, which is 2 by default but will also change in different simulation results considering distinct network conditions. We consider the default data availability is 10%, which means there will be one node storing the data needed by container i in every ten nodes on average. In the following Fig. 4, we will show the influence of changing data availability to performance of different algorithms. As mentioned in Section III, the ratio of operating cost to delay is important for multi-objective problems with inconsistent units. By default, we consider the operating cost of edge node γ_n is uniformly distributed in range $[0, 40]$. We will further evaluate the effects of different cost-delay ratios to our algorithm in Fig. 6. Similarly, the cloud cost parameter p is set to 0.5 by default, which means that operating the same container incurs on average half the cost compared to that at the edge. The value will also be tuned and evaluated in Fig. 6.

C. Baselines

In the simulations, we compare the ODCS algorithm with three typical container scheduling strategies as follows.

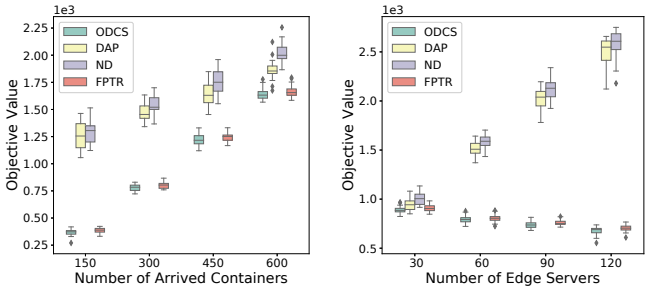
Data-aware placement algorithm (DAP): This baseline algorithm derives from the greedy container scheduling algorithm that is widely adopted by container orchestration systems like the Kubernetes [30]. The original algorithm assigns each condition of a server node a priority score and schedules the next arriving container on the node with the highest score. To acclimate the algorithm to edge environments for data-intensive applications, related work, e.g., [11], formulate the DAP algorithm by adding scores to evaluate the transmission distance between the edge node and data location. However, DAP does not take multiple data locations, detailed data flow routing, or operating costs of the serverless edge computing platform into consideration.

Near-data placement algorithm (ND): The ND algorithm is designed based on the fact that many edge-native applications tend to minimize the data transmission delay. Hence, the algorithm will deploy the next arriving container on the capable node, i.e., the node with sufficient resources, with the lowest transmission delay to the corresponding data location.

First placement then routing algorithm (FPTR): Instead of jointly dealing with container placement and flow routing, the FPTR algorithm first deploys the arriving container to the capable node with the lowest execution delay and operating cost. It then searches for the data location with the lowest transmission delay along the data flow routing path.

D. Simulation Results

In this section, we utilize box plots to show details of the simulation results and each box contains objective values of 20 independent simulations. We first present the performance of our ODCS algorithm and other baselines when the sufficiency of edge resources changes in Fig. 3. It is clear that an increasing number of arrived containers leads to a higher objective value with 60 nodes in the network as shown in Fig. 3(a). Nevertheless, our ODCS algorithm, which has a comprehensive consideration of flow routing and operating costs, always performs much better than the baselines without such features, e.g., the DAP algorithm. Such an advantage preserves even when the number of containers reaches 600, which means that the network is heavily loaded with the total resource consumption of containers comparable to the total capacity of all edge nodes. We can also draw from Fig. 3(a) that jointly handling the container placement and data flow routing as our algorithm does always achieve lower objective values compared to solving the two problems separately by the FPTR algorithm.

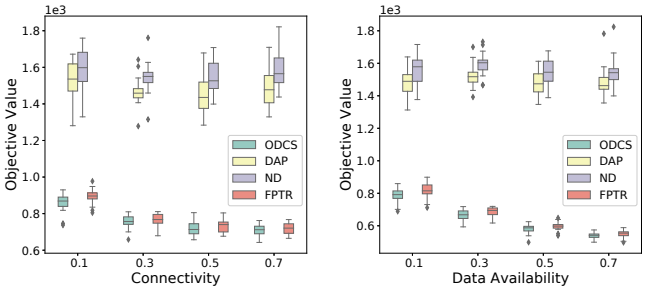


(a) (b)

Fig. 3. Performance of the ODCS algorithm and baselines under different resource sufficiency. (a) presents the comparison of the algorithms with fixed edge nodes and an increasing number of containers. (b) shows the performance with fixed containers and an increasing number of nodes.

Fig. 3(b) further shows the performance of the algorithms with in total of 300 containers deployed on an increasing number of nodes. Besides conclusions similar to those drawn from Fig. 3(a), we also observe that ignoring the operating cost of serverless computing on edge nodes may lead to even worse performance when the number of available edge nodes increases. This is because such algorithms like DAP and ND may initialize unnecessary edge devices for serverless computing. The proposed ODCS algorithm in this paper totally avoids this drawback and always achieves better performance with more sufficient edge resources.

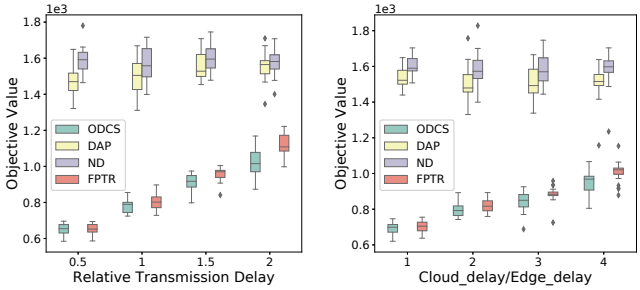
We then utilize Fig. 4(a) to evaluate the performance of different algorithms when the topology of the edge network varies. There are in total 60 nodes and 300 upcoming containers, the rest settings are the same as those in Fig. 3. There is an obvious performance improvement for the ODCS and FPTR algorithms when the connectivity of the edge network increases. We infer that such an enhancement is caused by the availability of more data flow routing choices with lower transmission delay. On the other hand, there is no such performance improvement in the DAP and ND algorithms due to their ignorance of detailed flow routing. As discussed in Section I, one advantage of the ODCS algorithm is that it respects that data could be available at multiple locations and picks the best one for objective function optimization. We thus



(a) (b)

Fig. 4. Performance of the ODCS algorithm and baselines in serverless edge computing networks with heterogeneous topology and data availability. (a) shows the simulation results of different algorithms when the connectivity of the edge network increases. (b) demonstrates corresponding results when the number of data copies grows.

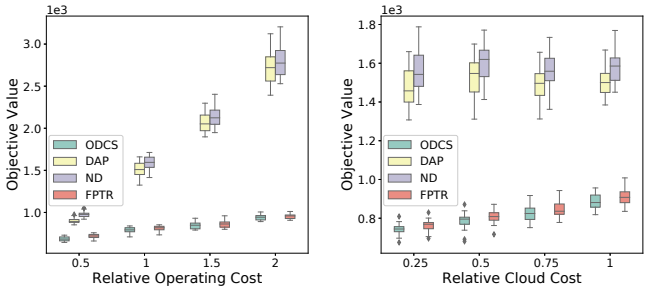
present the comparison of it to the baselines with different data availability in Fig. 4(b). We observe from the figure that the objective value of the ODCS algorithm keeps decreasing when the copies of data for containers increases in the edge network. In addition, ODCS still has the best performance compared to the baselines even when the data availability is low.



(a) (b)

Fig. 5. Performance of the ODCS algorithm and baselines with increasing delays. (a) presents the simulation results with growing transmission delays. (b) shows the results when the ratio of the edge-cloud transmission delay to the edge-edge transmission delay increases.

The geo-distribution and heterogeneity of edge networks are also reflected in the diverse transmission delays among edge nodes. We multiply the default transmission delay in the settings by a factor changing from 0.5 to 2 and show corresponding simulation results in Fig. 5(a). It is worth noting that the performance declines of the ODCS and the FPTR algorithms are more prominent than the other two baselines DAP and ND. The reason is that the latter two algorithms always tend to deploy containers on nodes with the lowest transmission delay to data, thus largely offsetting the influence of transmission delay increment. The former two algorithms, on the other hand, need to balance the operating costs and the delays. Nevertheless, ODCS and FPTR still outperform the other two baselines even with large transmission delays. Furthermore, the exceeding of our ODCS to the FPTR algorithm is more obvious under large transmission delays, emphasizing the advantages of solving container placement and flow routing jointly. Similarly, we can infer from Fig. 5(b) that the ODCS still achieves the best performance even if the transmission delay between edge and cloud is on average four times larger than that among edge nodes.



(a) (b)

Fig. 6. Performance of the ODCS algorithm and baselines under different operating costs. (a) shows the algorithms' performance when the ratio of the operating cost to the default value in the settings increases from 0.5 to 2. (b) shows corresponding results when p change from 0.25 to 1.

In the end, we present the effect of different operating costs on the performance of the OCDS algorithm and other baselines in Fig. 6. Although the objective values of all algorithms increase with the growth of the relative operating cost as shown in Fig. 6(a), the increments of ODCS and FPTR are much milder than those of the other two baselines. Besides, the ODCS algorithm remains the leading performance even when the operating cost is relatively small compared to the delays, which shows the superiority of the proposed algorithm in solving the multi-objective problem. Fig. 6(b) further presents the simulation results when the relative cloud operating cost, i.e., the parameter p, varies while the operating cost of each edge node remains the same. We can conclude from the figure that our ODCS can orchestrate containers flexibly among edge nodes and the cloud for the best performance. It will achieve a lower objective value if operating a container on the cloud is relatively cheaper but still keeps the best performance compared with other baselines when cloud resources are expensive.

VI. Conclusion

In this paper, we mainly focus on extending the serverless computing paradigm from the cloud to the network edge to facilitate data-intensive applications. We demonstrate the challenges of serverless edge computing introduced by the geo-distribution and heterogeneity of edge environments. We propose an online data-aware container scheduling algorithm to address the aforementioned challenges, which deals with container placement and data flow routing simultaneously. We prove that the proposed algorithm achieves local optimum for each arriving container request and its global result also preserves a theoretical bound to the optimal offline solution. We further conduct extensive simulations based on real-world experiment-driven data to show that our algorithm performs better than existing baselines.

Acknowledgments

This work was supported in part by the National Science Foundation under grant number CCF-2230620, CCF-2046444, CNS-2146909, CNS-2106027, and CNS-2214980.

REFERENCES

[1] I. Baldini, P. Cheng, S. J. Fink, N. Mitchell, V. Muthusamy, R. Rabbah, P. Suter, and O. Tardieu, "The serverless trilemma: Function composition for serverless computing," in *2017 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*. ACM, 2017, pp. 89–103.

[2] E. Jonas, J. Schleier-Smith, V. Sreekanti, C.-C. Tsai, A. Khandelwal, Q. Pu, V. Shankar, J. Carreira, K. Krauth, N. Yadwadkar *et al.*, "Cloud programming simplified: A berkeley view on serverless computing," *arXiv preprint arXiv:1902.03383*, 2019.

[3] P. Castro, V. Ishakian, V. Muthusamy, and A. Slominski, "Serverless programming (function as a service)," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 2658–2659.

[4] L. Baresi, D. Filgueira Mendonça, and M. Garriga, "Empowering low-latency applications through a serverless edge computing architecture," in *European Conference on Service-Oriented and Cloud Computing*. Springer, 2017, pp. 196–210.

[5] A. Glikson, S. Nastic, and S. Dustdar, "Deviceless edge computing: extending serverless computing to the edge of the network," in *2017 The 10th ACM International Systems and Storage Conference*. IEEE, 2017, pp. 1–1.

[6] L. Baresi and D. F. Mendonça, "Towards a serverless platform for edge computing," in *2019 IEEE International Conference on Fog Computing*. IEEE, 2019, pp. 1–10.

[7] C. Cicconetti, M. Conti, and A. Passarella, "A decentralized framework for serverless edge computing in the internet of things," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 2166–2180, 2020.

[8] Y. Xiong, Y. Sun, L. Xing, and Y. Huang, "Extend cloud to edge with kubeedge," in *2018 IEEE/ACM Symposium on Edge Computing*. IEEE, 2018, pp. 373–377.

[9] C. Cicconetti, M. Conti, and A. Passarella, "An architectural framework for serverless edge computing: design and emulation tools," in *2018 IEEE international conference on cloud computing technology and science (CloudCom)*. IEEE, 2018, pp. 48–55.

[10] S. Wang, Y. Hu, and J. Wu, "Kubeedge. ai: Ai platform for edge devices," *arXiv preprint arXiv:2007.09227*, 2020.

[11] T. Rausch, A. Rashed, and S. Dustdar, "Optimized container scheduling for data-intensive serverless edge computing," *Future Generation Computer Systems*, vol. 114, pp. 259–271, 2021.

[12] S. Eismann, J. Scheuner, E. Van Eyk, M. Schwinger, J. Grohmann, N. Herbst, C. L. Abad, and A. Iosup, "A review of serverless use cases and their characteristics," *arXiv preprint arXiv:2008.11110*, 2020.

[13] M. S. Aslanpour, A. N. Toosi, C. Cicconetti, B. Javadi, P. Sbarski, D. Taibi, M. Assuncao, S. S. Gill, R. Gaire, and S. Dustdar, "Serverless edge computing: vision and challenges," in *2021 Australasian Computer Science Week Multiconference*, 2021, pp. 1–10.

[14] K3S, "K3s: Lightweight Kubernetes," Online, https://k3s.io/.

[15] Kubernetes, "Kubelet," Online, https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet/.

[16] Microsoft, "Azure Functions," Online, https://azure.microsoft.com/en-us/services/functions/.

[17] Google, "Cloud Functions," Online, https://cloud.google.com/functions/.

[18] Amazon, "Lambda," Online, https://aws.amazon.com/lambda/.

[19] IBM, "Openwhisk," Online, https://openwhisk.apache.org/.

[20] I. E. Akkus, R. Chen, I. Rimac, M. Stein, K. Satzke, A. Beck, P. Aditya, and V. Hilt, "{SAND}: Towards {High-Performance} serverless computing," in *2018 Usenix Annual Technical Conference*. USENIX, 2018, pp. 923–935.

[21] W. Lloyd, S. Ramesh, S. Chinthalapati, L. Ly, and S. Pallickara, "Serverless computing: An investigation of factors influencing microservice performance," in *2018 IEEE International Conference on Cloud Engineering*. IEEE, 2018, pp. 159–169.

[22] S. K. Mohanty, G. Premsankar, M. Di Francesco *et al.*, "An evaluation of open source serverless computing frameworks." in *CloudCom*, 2018, pp. 115–120.

[23] L. Feng, P. Kudva, D. Da Silva, and J. Hu, "Exploring serverless computing for neural network training," in *2018 IEEE 11th international conference on cloud computing*. IEEE, 2018, pp. 334–341.

[24] A. Jangda, D. Pinckney, Y. Brun, and A. Guha, "Formal foundations of serverless computing," *ACM on Programming Languages*, vol. 3, no. OOPSLA, pp. 1–26, 2019.

[25] L. Patterson, D. Pigorovsky, B. Dempsey, N. Lazarev, A. Shah, C. Steinhoff, A. Bruno, J. Hu, and C. Delimitrou, "Hivemind: a hardware-software system stack for serverless edge swarms," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 800–816.

[26] Amazon, "AWS IoT Greengrass," Online, https://aws.amazon.com/greengrass/.

[27] S. Nastic, T. Rausch, O. Scekic, S. Dustdar, M. Gusev, B. Koteska, M. Kostoska, B. Jakimovski, S. Ristov, and R. Prodan, "A serverless real-time data analytics platform for edge computing," *IEEE Internet Computing*, vol. 21, no. 4, pp. 64–71, 2017.

[28] Docker, "Docker," Online, https://www.docker.com/.

[29] OpenFaaS, "OpenFaaS: Serverless Functions Made Simple," Online, https://www.openfaas.com/.

[30] Kubernetes, "Kubernetes," Online, https://kubernetes.io/.

[31] L. Pan, L. Wang, S. Chen, and F. Liu, "Retention-aware container caching for serverless edge computing," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 2022, pp. 1–10.

[32] Y. Liu, X. Shang, and Y. Yang, "Joint sfc deployment and resource management in heterogeneous edge for latency minimization," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 8, pp. 2131–2143, 2021.

[33] Y. Mao, X. Shang, and Y. Yang, "Provably efficient algorithms for traffic-sensitive sfc placement and flow routing," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 2022, pp. 950–959.

[34] X. Shang, Y. Huang, Z. Liu, and Y. Yang, "Reducing the service function chain backup cost over the edge and cloud by a self-adapting scheme," *IEEE Transactions on Mobile Computing*, 2021.

[35] Y. Mao, X. Shang, and Y. Yang, "Joint resource management and flow scheduling for sfc deployment in hybrid edge-and-cloud network," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 2022, pp. 170–179.

[36] Raspberry, "raspberry pi 4," Online, https://www.raspberrypi.com/products/raspberry-pi-4-model-b/.

[37] NVIDIA, "Jetson xavier nx," Online, https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-nx/.