Towards Cost-Efficient Edge Intelligent Computing With Elastic Deployment of Container-Based Microservices

PENG ZHAO\textsuperscript{1}, (Member, IEEE), PEIZHE WANG\textsuperscript{2}, XINYU YANG\textsuperscript{2}, (Member, IEEE), AND JIE LIN\textsuperscript{2}, (Member, IEEE)
\textsuperscript{1}School of Computer Science and Technology, Xi’an Jiaotong University, Xi’an 710049, China
\textsuperscript{2}School of Software Engineering, Xi’an Jiaotong University, Xi’an 710049, China
Corresponding author: Xinyu Yang (yxyphd@mail.xjtu.edu.cn)

This work was supported in part by the National Key Research and Development Program of China under Grant 2017YFB1010004.

ABSTRACT With the tremendous growth of the Internet of Things (IoT), big data, and artificial intelligence (AI), the edge computing-based service paradigm has been introduced to meet the increasing demand of applications. To provide efficient computing services at the network edge, the algorithms and applications are generally deployed based on the container-based microservice strategy, which significantly impacts the system efficiency and QoS. Considering the fundamental system uncertainties, including the dynamic workload and service rate, we investigate how to minimize the long-term system cost through the elastic microservice deployment in this paper. To this end, we formulate the container-based microservice deployment as a stochastic optimization problem to minimize the system cost while maintaining the system QoS and stability. We develop a cost-aware elastic microservice deployment algorithm to solve the formulated problem, which balances the tradeoff between system cost and QoS. Our algorithm makes the real-time decisions based on current queue backlogs and system states without predicting the future knowledge. Finally, we conduct the theoretical analysis and extensive simulations based on data traces from the ResNet-50 model-based visual recognition application. The results demonstrate that our algorithm outperforms the baseline strategies with respect to the system cost, queue backlogs, and the number of Pod replicas.

INDEX TERMS Edge computing system, microservices, elastic deployment, system cost, container.
the host OS, and each virtual machine has an independent OS. The services are provided by deploying applications in the VM. Operating system-level virtualization technology is also known as container-based virtualization. It provides OS kernel to each container in the form of a process through the container engine, which is isolated from each other. The OS kernel manages the resource allocation of each container. Since VM-based virtualization isolates host machine resources at the hardware level and requires additional user OS for supporting application deployments, more resources will be occupied, and the longer time is required to create and launch the VM. Instead, new processes will be created directly in the host OS kernel by deploying a new container in container-based virtualization. Thus, considering the limited resource and strict latency requirement in edge computing, it is necessary to dynamically provide flexible services through the container-based virtualization to meet the computing requirements and efficient resource utilization [10]. Moreover, the microservice architecture is generally used to deploy services in container-based virtualization technology. Because the microservice architecture could decouple the complex applications into weak-linked, light components, with each component running a microservice that can be shared when the microservice overlap exists in applications, and each component can be replaced and updated without involving other components [11]. Therefore, container-based microservices not only meet the resource and latency requirements of service deployment in edge computing but also improves the elasticity and scalability of service performance [12].

In practice, the service deployment strategy can be divided into active and passive approaches [13]. In the former way, services are pre-deployed using excessive or sufficient resources to achieve a predictable deployment plan [12], [14]–[17]. For example, Calheiros et al. in [15] proposed the task load prediction and the active resources scheduling strategy based on the autoregressive integral sliding average model. Bhattacharjee et al. in [17] presented a containerized serviceless computing framework for deploying the pre-trained deep learning models in the cloud environment. The proposal predicted the resource requirements ahead of the incoming requests, and a forecast-aware scheduling mechanism, which improves resource utilization while preventing physical resource exhaustion. The latter way designs the deployment plans passively based on the system models constructed based on the application principles, historical data, or experts experiences [18]–[21]. For example, Zhu et al. in [19] proposed a dynamic resource supply algorithm based on a multi-input and multi-output feedback control model, in which the dynamic deployment adjustment under the constraints of resource budget and time-limit is realized for maximizing the service quality. Rossi et al. [21] proposed a reinforcement learning based strategy to dynamically control the container deployment and allocate the computing resource to the application. Moreover, this paper also solved the container placement problem in a geographically distributed environment through integer linear programming and network awareness.

Nonetheless, these existing deployment methods are not flexible enough in the face of random system workload, and cannot be flexibly deployed by adjusting resources in real time according to the dynamic system states. First, planning the service and resource deployment in advance will lead to redundant idle services and increase the resource waste and power consumption, because the workload and service deployment are independent of each other. Moreover, if the service and resources are adjusted actively by the prediction model, the efficiency may be limited by the accuracy of the prediction model, resulting in the redundancy or shortage of resources and finally affecting the Quality of Service (QoS).

In this paper, we propose a Cost-aware Microservice Elastic Deployment (CMED) algorithm, which dynamically adjusts the number of Pod replicas according to the real-time workload to minimize the system cost while ensuring the system stability, without predicting the value of any unpredictable system parameters. The main contributions of this paper are summarized as follows:

1) We established fine-grained system models to capture the characteristics of edge computing and service deployment in containers, and formulated the microservice deployment as an optimization problem mathematically. In particular, we first present the system architecture of container-based microservice deployment at the network edge. Then, the container-based microservice deployment is described as a stochastic optimization problem by controlling the number of Pod replicas to minimize the system cost while maintaining system stability.

2) A stochastic optimization method is proposed to solve the microservice elastic deployment problem. More specifically, the Lyapunov optimization framework is used to balance the system cost and QoS while achieving system stability. As a result, a cost-aware microservice elastic deployment (CMED) algorithm is developed to make the decision based on the measurable system parameters without qualifying the dynamic system states in advance.
3) The performance of the CMED algorithm was evaluated by theoretical analysis and extensive simulations. In particular, the performance bounds on system cost and queue backlogs are analyzed and proved theoretically. Meanwhile, a large number of simulations were conducted to verify the performance based on data traces, which are generated from the ResNet-50 model-based visual recognition application in a realistic physical environment. The results show that our proposal outperforms the baseline algorithms in balancing the system cost and QoS.

The rest of this paper is organized as follows. In Section II, we provide a brief overview of the related research. In Section III, we present the system model and describe the service deployment problem mathematically. In Section IV, we introduce the CMED algorithm to solve the optimization problem. In Section V, we evaluate the performance of the algorithm through experiments and comparisons. Finally, we summarize this paper in Section VI.

II. RELATED WORK

There has been a lot of research on the service deployment problem to provide efficient and reliable services. In the following, we conduct a literature review of the existing work related to our study. In general, existing service deployment strategies can be divided into active strategies based on predictable models and passive deployment strategies based on the dynamic system information [13].

The active deployment strategy generally uses the historical information about the system to train or build a prediction model, and make the deployment prediction based on the predicted system workload and dynamic parameters. Finally, the deployment plan is adjusted according to the prediction results to achieve the matching of resource and load requirements. For example, Wan et al. in [12] allocate resources to a set of microservice containers in the application, set idle microservices to sleep and wake up when needed, and migrated or amplified other physical machines for performance supplement when the deployed service performance could not meet the requirement. Alam et al. in [14] present the real-time task switching scheme for microservices at different network levels by pre-loading Docker image into nodes. Calheiros et al. in [15] proposed the strategy for improving the QoS of cloud service through dynamic resource provision. The strategy estimate the future resource requirement of applications and allocate them in advance, which is achieved by a cloud workload prediction module based on the autoregressive integrated moving average (ARIMA) model. Bhattacharjee et al. in [17] proposed a methodology to estimated the required resource based on the workload forecasting and latency bound, and then presented a mechanism using the serverless paradigm to allocate resources proactively based on the estimated resource requirement and current system states.

The passive deployment strategy generally conducts the real-time service deployment and resource allocation in combination with real-time system states. This type of approach generally combines the mathematical system model and certain optimization methods to achieve the matching of resources and workload requirements. For example, Lin et al. in [18] determined the deployment plan by establishing a model, and then dynamically adapted the number of active servers to match the current workload for power cost saving. Zhu et al. in [19] established a dynamic resource supply algorithm based on a multi-input and multi-output feedback control model to dynamically adjust the resource deployment under the constraints of resource budget and time-limit for maximizing the QoS. Zhao et al. in [20] studies the existing auto-scaling strategy of Kubernetes and proposes an auto-scaling capacity expansion strategy, which can optimize the response delay caused the time required for Pod initialization. This strategy uses a combination of empirical modal decomposition and ARIMA model to adjust the number of Pods in the Kubernetes cluster. Rossi et al. in [21] studied the container deployment problem in geo-distributed computing environments. In particular, a reinforcement learning based solution was proposed to control the horizontal and vertical deployment of the containers, and the container placement problem was also addressed by solving an integer linear programming or using a network-aware heuristic algorithm.

Different from the existing work, the microservice deployment algorithm proposed in this paper makes decisions based on the real-time system states without predicting the dynamic task arrival and service rate. In particular, the randomness caused by the system dynamics overtime can be well modeled by abstracting system resources and services into a queue. Finally, the problem of service deployment and resource allocation are combined into one algorithm for improving the overall efficiency.

III. SYSTEM MODEL AND PROBLEM FORMULATION

In this section, we first describe the functional system architecture of edge computing service for AI applications. Then, the system models are introduced to describe the system mathematically. Using these models, we finally formulate the problem of deploying microservices in a cost-efficient system way as a constrained optimization problem.

A. SYSTEM ARCHITECTURE

We consider a service paradigm for deploying various AI applications in a container-based edge computing service system. As shown in Figure 2, the system is composed of several computing devices at the network edge in different geographical locations. The system is re-divided into several computing nodes by uniformly integrating and scheduling the resources composed of geographically distributed infrastructures. The AI applications can be orchestrated by a set of microservices that are developed and deployed with containers on these nodes for providing services to users. In particular, Kubernetes can be used to cooperate with Docker as
We assume that the system consists of a computing system to optimize the number of deployed Pod replicas of the edge and resource utilization. In this article, we will focus on how the number of Pod replicas will become the key of affecting QoS and resource utilization. Thus, the appropriate adjustment of the replicas can be adjusted to provide the elastic service.

As we have introduced above, the user’s tasks are performed by the deployed multiple Pod replicas of the practical system. As we have introduced above, the user’s tasks are performed by the deployed multiple Pod replicas of the practical system. In order to match the dynamic workload and improve resource utilization, the number of deployed Pod replicas should be dynamically adjusted according to the real-time system states. Denote the number of replicas deployed in the system in time slot $t$ as $n(t)$. Due to the physical limitations of the entire system, there must exist a maximum number of deployable replicas, denoted as $n_{\text{max}}$. That is, $n(t)$ should satisfy $0 \leq n(t) \leq n_{\text{max}}$.

Here, we define the service rate of each replica as the number of tasks processed during each time slot. Denote the service rate of the $i_{\text{th}}$ replica in time slot $t$ by $r_i(t)$. Notice that $r_i(t)$ is a random parameter due to the dynamic system resource and the difference between each task. Nonetheless, there always exists a maximum amount of task processed by each replica (denoted as $r_{\text{max}}$) and satisfies the constraint of $1/T_{\text{max}} \leq r(t) \leq r_{\text{max}}$, where $T_{\text{max}}$ is the maximum response delay allowed by the task. Because when the number of replicas is sufficient, the system can directly process all arrived task requests, then the time required for processing the task in a single replica should also meet the requirement of maximum response delay.

According to the above description, we can get the number of requests processed by the system $\mu(t)$, the number of Pod replicas $n(t)$, and the number of requests processed by each replica $r_i(t)$, should satisfy the following constraints:

$$\mu(t) = \sum_{i=1}^{n(t)} r_i(t), \quad 0 \leq n(t) \leq n_{\text{max}}, \quad \frac{1}{T_{\text{max}}} \leq r(t) \leq r_{\text{max}}.$$

### 1) WORKLOAD MODEL

For simplicity, we consider only one type of task served by the system for massive users. Notice that the system model can be easily extended to the scenario of multiple services with different types. In time slot $t$, we suppose that $M$ users have initiated requests to the system, the number of tasks initiated by the $i_{\text{th}}$ user in the time slot is $\lambda_i(t)$, then the task arrival of all users can be expressed as $\lambda(t) = \sum_{i=1}^{M} \lambda_i(t)$. Accordingly, the average workload of the system (denoted as $\lambda$) can be defined as the time average task arrivals of all users. In practice, the number of randomly arrived tasks in any time slot cannot be infinite due to constraints on network bandwidth and client devices. Thus, we assume that there always exists an upper limit $\lambda_{\text{max}}$ that satisfies $0 \leq \lambda(t) \leq \lambda_{\text{max}}$ in all time slots, i.e., the following constraints must be held for any time slot $t$:

$$\lambda(t) = \sum_{i=1}^{M} \lambda_i(t), \quad 0 \leq \lambda(t) \leq \lambda_{\text{max}}.$$

### 2) COMPUTATION SERVICE MODEL

As we have introduced above, the user’s tasks are performed by the deployed multiple Pod replicas of the practical system. In order to match the dynamic workload and improve resource utilization, the number of deployed Pod replicas should be dynamically adjusted according to the real-time system states. Denote the number of replicas deployed in the system in time slot $t$ as $n(t)$. Due to the physical limitations of the entire system, there must exist a maximum number of deployable replicas, denoted as $n_{\text{max}}$. That is, $n(t)$ should satisfy $0 \leq n(t) \leq n_{\text{max}}$.

Under the service deployment with the QoS guarantee, if the resources for each microservice can be reasonably allocated, the unnecessary resource consumption will be avoided, and consequently reduce the system cost and improve the resource utilization. Thus, the appropriate adjustment of the number of Pod replicas will become the key of affecting QoS and resource utilization. In this article, we will focus on how to optimize the number of deployed Pod replicas of the edge computing system.

### B. SYSTEM MODEL

We assume that the system consists of $N$ edge computing devices, denoted as $D = \{D_1, D_2, \ldots, D_N\}$, where each computing device has fixed computing resources. The system’s operation is divided into multiple consecutive time slots, which is represented by $t$, and the length of each time slot matches the timescale for the system to perform a series of operations. The specific modeling of each component is introduced as follows.
3) DYNAMIC QUEUE MODEL

As shown in Figure 3, the service engine in our system is modeled as a dynamic queue, which maintains the unfinished task requests. Here, we denote the backlog of unprocessed tasks in the queue is $Q(t)$ at each time slot $t$. During any time slot $t$, users will generate some new service requests randomly (i.e., $\lambda(t)$), and a certain number of requests will be completed by the deployed microservices (i.e., $\mu(t)$). Then, the queue dynamics of $Q(t)$ and evolution obeys the following equation:

$$Q(t + 1) = \max\{Q(t) - \mu(t), 0\} + \lambda(t),$$  \hspace{1cm} (6)

where $Q(t + 1)$ is the queue backlog at the beginning of next time slot $t + 1$. We can find that if the number of processed tasks in time slot $t$ (i.e., service rate $\mu(t)$) is less than the number of arrived tasks $\lambda(t)$, new tasks will pile up in the queue; conversely, when the service rate is greater than the task arrival, the tasks in the queue will gradually decrease.

Theoretically, we don’t expect the queue will grow infinitely and cause the system to crash. In other words, the time average of backlog tasks in the queue will not be infinite. Then, the $Q(t)$ must satisfy the following relationship:

$$\overline{Q(t)} := \lim_{t \to \infty} \sup T \sum_{t=0}^{T-1} E[|Q(t)|] < \infty. \hspace{1cm} (7)$$

In fact, the relationship in (7) is just the most basic assumption for guaranteeing the system stability in theory, which is conducted based on the fundamental Lyapunov stability. In the practical system, the limitation of queue backlog $Q(t)$ should be restricted to some constant value due to the limited buffer capacity of edge devices.

4) SYSTEM COST MODEL

Now, we introduce the system cost caused in the process of deploying Pod replicas. Considering the limited system resource, the deployment of Pod replicas is always accompanied by the corresponding system cost. Generally speaking, the system cost can be divided into the following two parts:

a: Deployment Physical Cost

When an additional Pod replica is deployed, the system needs to allocate certain system resources to the replica, which will increase the system consumption and increase the physical cost in service providing. In practice, the physical cost can be considered as the monetary cost of buying the resource for deploying the replica, or the energy consumption of running the replica, etc. For any time slot $t$, we denote such physical cost as $c_p(t)$. For the case in this paper, because only one service is involved in the system and the replica is deployed in the system using container-based virtualization technology, it can be considered that the overall physical cost and the number of Pod replicas $n(t)$ have a linear relationship; that is, the following formula holds:

$$c_p(t) = \rho \cdot n(t), \hspace{1cm} (8)$$

where $\rho$ is the system cost required for each replica.

b: Deployment Operation Cost

For any time slot $t$, if the deployment plan decides to increase the number of replicas compared to the previous time slot $t - 1$, the system needs some additional cost to perform the operations of allocating the additional resources, constructing, configuring, and launching the new replicas. Generally speaking, such operation cost of adding additional replicas could include the cost of performing these operations and the delay cost before the new replicas are ready for providing services. Notice that the delay cost of deploying a new replica varies greatly from different specific applications, and may vary from a few seconds to several minutes [18]. Instead, when the number of replicas is reduced, there is no extra operation cost.

For any time slot $t$, we denote the operation cost as $c_o(t)$. Then, the relationship between the operation cost $c_o(t)$ and the corresponding number of replicas $n(t)$ satisfies the following relationship:

$$c_o(t) = \tau \cdot \max\{n(t) - n(t - 1), 0\}, \hspace{1cm} (9)$$

where $n(t - 1)$ is the number of Pod replicas in time slot $t - 1$; $\tau$ is the operation cost required for each new replica deployment.

To sum up, we define the overall system cost of deploying microservices during each time slot as a combination of the physical cost and the operation cost. Denote the overall system cost of service deployment in time slot $t$ as $C(t)$, then we have:

$$C(t) = \omega_1 \cdot c_p(t) + \omega_2 \cdot c_o(t), \hspace{1cm} (10)$$

where $\omega_1$ and $\omega_2$ are two non-negative weight parameters for the personalized tradeoff between the physical cost and the operation cost.

C. PROBLEM FORMULATION

In this paper, we consider minimizing the total system cost by dynamically controlling the number of Pod replicas while ensuring the service quality and system stability. In fact, fewer Pods mean smaller system cost. However, this does not mean that the number of deployed Pod replicas should be reduced, because because the processing capacity of the system will be weak and may cause a long queue backlog and the poor QoS when the number of Pod replicas is insufficient. If the number of Pods is kept at an average level, redundancy and insufficient computing power may occur at some
moments. When the number of Pod replicas is changed, if the number of Pod replicas is increased, the system will spend a certain amount of cost (i.e., delay) to deploy the Pod replicas. Therefore, we should make an optimal decision on service deployment by simultaneously considering the system cost and service quality based on the dynamic system states.

According to the system model mentioned above, we formulate the microservice deployment of edge computing as the following optimization problem of minimizing the total system cost subject to the constraints on request arrival, system capacity, stability. Mathematically, we have the following constrained optimization problem:

\[
\min_{C_{av}} \quad L(t+1) - L(t)
\]

s.t. (1), (2), (3), (4), (5), and (7).

The strategy which is feasible and can satisfy the optimization problem (11) is called a stable strategy, and \( C_{av} \) is used to represent the time average system cost. For this optimization problem, we need to find a relatively stable strategy and use this to control the deployment of Pod replicas. In practice, the minimum system cost can be achieved only in the system with delay-tolerant tasks. In reality, there always exists an acceptable response time (i.e., patience) for users. Response time violation may consequently dent the system’s appeal to clients, and thus reduce its competitiveness in the market. Therefore, the practical control objective is to achieve a tradeoff between the minimum system cost and QoS, while stabilizing the queues of the system.

IV. COST-AWARE MICROSERVICE ELASTIC DEPLOYMENT

To solve the optimization problem defined above under the system with unpredictable random parameters, we consider the concept of Lyapunov optimization theory, in which we can achieve optimal control of the system by using dynamic coefficients. In this paper, the Lyapunov optimization framework is used to describe the model of task queue backlogs, and the optimal decision is made by using the Lyapunov function based on the real-time system states. Because the elastic decision can be made greedily just based on the current system states without considering the future knowledge of stochastic factors. In the following, the elastic microservice deployment algorithm will be presented in detail.

A. LYAPUNOV OPTIMAL FRAMEWORK

In order to describe the current task queue backlog, the Lyapunov function \( L(t) \) is defined to represent the task accumulation at time slot \( t \), as follows:

\[
L(t) := \frac{1}{2} Q(t)^2.
\]

In addition, Lyapunov drift is used to represent the degree of the queue change in the adjacent time slot; that is, the difference between the value of the Lyapunov function in the next time slot and current time slot. Here, the Lyapunov drift is defined as follows:

\[
\Delta(t) := L(t+1) - L(t).
\]

To quantify the impact of the system cost on the queue stability, \( V \) is introduced as a control coefficient. The parameter \( V \) is a non-negative parameter that is chosen as the desired weight to affect the performance tradeoff. To stabilize the task queue while minimizing the time average system cost \( C_{av} \), the drift-plus-penalty expression \( f(t) \) is introduced by adding the Lyapunov drift and the system cost with the control coefficient \( V \). It’s not difficult to find that the algorithms can be designed to take control actions that greedily minimize a bound on the following drift-plus-penalty expression of each slot \( t \).

\[
f(t) = \Delta(t) + V \cdot C(t).
\]

The key of solving the above problem is to derive an upper bound on (14). Based on the constraints on task arrival, number of Pod replicas, etc., we can easily derive the following upper bound:

\[
\mathbb{E}[f(t)|Q(t)]
\]

\[
\leq B + \mathbb{E}[V \cdot C(t)|Q(t)] + \mathbb{E}[Q(t) \cdot \lambda(t)]|Q(t)\]

\[
\leq B + \mathbb{E}[Q(t) \cdot \lambda(t)|Q(t)] + \mathbb{E}[V \cdot C(t)]
\]

\[
+ \mathbb{E}\left[ \frac{1}{2} \mu(t)^2 - \mu(t) \cdot [Q(t) + \lambda(t)]|Q(t) \right].
\]

The detailed proof is presented as follows.

**Proof:** Considering the facts that \( \max\{Q(t)-\mu(t), 0\} \leq [Q(t)-\mu(t)]^2 \) and \( \max\{Q(t)-\mu(t), 0\} \leq Q(t) \), we can derive that (13) has the following bound:

\[
\Delta(t) \leq \frac{1}{2} \frac{1}{2} Q(t)^2 - \frac{1}{2} Q(t)^2
\]

\[
\leq B(t) + \frac{1}{2} \mu(t)^2 - \mu(t) \cdot [Q(t) + \lambda(t)] + \lambda(t) Q(t),
\]

where \( B(t) = \frac{1}{2} \lambda(t)^2 \). For \( B(t) \), it is easy to prove that the following relationship exists:

\[
B(t) \leq \frac{1}{2} \lambda_{max}^2 = B.
\]

Then, (16) and (17) can be used to derive that the Lyapunov drift function satisfies the following relationship:

\[
\Delta(t) \leq B(t) + \frac{1}{2} \mu(t)^2 - \mu(t) \cdot [Q(t) + \lambda(t)] + \lambda(t) Q(t).
\]

By adding the Lyapunov drift function and the penalty function, we can finally get the following expression:

\[
\Delta(t) + \mathbb{E}[V \cdot C(t)|Q(t)]
\]

\[
= \mathbb{E}[f(t)|Q(t)]
\]

\[
\leq B + \mathbb{E}[Q(t) \cdot \lambda(t)|Q(t)] + \mathbb{E}[V \cdot C(t)]
\]

\[
+ \mathbb{E}\left[ \frac{1}{2} \mu(t)^2 - \mu(t) \cdot [Q(t) + \lambda(t)]|Q(t) \right].
\]
the number of Pod replicas, so as to reach the previous bound of the drift-plus-penalty function to ensure its value is sufficiently small. More specifically, the system can just control the last two items to achieve the optimization object. The details of the algorithm will be presented in the next subsection.

B. ELASTIC MICROSERVICE DEPLOYMENT ALGORITHM

Based on the above theoretical analysis, the cost-aware microservices elastic deployment (CMED) algorithm is designed to schedule the service deployment by adjusting the number of replicas dynamically. For arbitrary control parameter $V$ and given system parameters (e.g., queue backlog $Q(t)$, the weights of cost $\omega_1$ and $\omega_2$, task arrival $\lambda(t)$, etc.), it is possible to design the optimized deployment strategy.

Because the operation cost only exists when the number of Pod replicas of current time slot $t$ is greater than that of the previous time slot $t-1$ (i.e., $n(t) \geq n(t-1)$), we first consider the situation that there is no operation cost (i.e., $c_o(t)$) in the problem, and then solve the corresponding optimization problem to get the optimal number of replicas. So, the optimization problem can be re-expressed as follows:

$$
\min_{n(t)} V_1 \cdot c_p(t) + \frac{1}{2} \mu(t)^2 - \mu(t)[Q(t) + \lambda(t)]
$$

s.t. $0 \leq n(t) \leq n_{\text{max}}$

$$
\mu(t) = \sum_{i=1}^{n(t)} r_i(t)
$$

$$
\frac{1}{T_{\text{max}}} \leq r(t) \leq r_{\text{max}}
$$

$$
c_p(t) = \rho \cdot n(t),
$$

where $V_1 = V \cdot \omega_1$.

If the optimal number of replicas $n(t)$ obtained by solving the problem in (20) is less than that of the previous time slot $n(t-1)$, i.e., $n(t) \leq n(t-1)$, then the system does not need to deploy additional new replicas in the incoming time slot $t$; as a result, the corresponding operation cost (i.e., $c_o(t)$) does not exist. In other words, $n(t)$ is the optimal solution. Conversely, we should consider the operation cost $c_o(t)$ in designing the service deployment algorithm. Thus, in the case of $n(t) > n(t-1)$, the optimization problem can be re-expressed as follows:

$$
\min_{n(t)} V_1 \cdot c_p(t) + V_2 \cdot c_o(t) + \frac{1}{2} \mu(t)^2 - \mu(t)[Q(t) + \lambda(t)]
$$

s.t. $0 \leq n(t) \leq n_{\text{max}}$

$$
\mu(t) = \sum_{i=1}^{n(t)} r_i(t)
$$

$$
\frac{1}{T_{\text{max}}} \leq r(t) \leq r_{\text{max}}
$$

$$
c_p(t) = \rho \cdot n(t)
$$

$$
c_o(t) = \tau \cdot [n(t) - n(t-1)],
$$

where $V_1 = V \cdot \omega_1$ and $V_2 = V \cdot \omega_2$.

By solving the above two optimization problems, the microservice deployment plan can be finally obtained. It can be found that the problems mentioned in (20) and (21) are both quadratic functions related to $n(t)$, which can be efficiently solved by taking the derivation of corresponding objective function. In the following, we introduce the details of the cost-aware microservice elastic deployment (CMED) algorithm based on the given system parameters, such as $V$, $\omega_1$, $\omega_2$, etc. Initially (i.e., $t = 0$), since there are no tasks arrival and the number of tasks backlog in the initialization queue is zero, the number of pod replicas deployed in the system $n(0)$ is initialized as zero as well. Then, the following operations are performed by the service deployment component in the system at every time slot $t$:

- **System States Observation**: The service deployment component observes the number of tasks arrived at the system $\lambda(t)$, current levels of queue backlog $Q(t)$, etc.

- **Microservice Deployment Plan Making**: Based on the given system parameters, the current system states, we first solve the optimization problem of (20) to get a temporary deployment plan, i.e., the number of replicas $n(t)$. If $n(t) \geq n(t-1)$, this temporary deployment plan is finally decided; otherwise, we solve the optimization problem of (21) to obtain the final deployment plan $n(t)$. Finally, the system deploys the corresponding Pod replica based on the final plan and serves the task requests.

- **Queue Backlog Updating**: According to the newly arrived tasks and processed tasks, the system updates its queue backlog.

The pseudocode of the CMED algorithm is presented in Algorithm (1). Obviously, the complexity of Algorithm (1) is $O(1)$. Notice that the performance of CMED is controlled by the parameter $V$, which could achieve the tradeoff between the average system cost and queue backlogs. The average queue backlog also reflects the system QoS in terms of latency. In the next section, we will conduct the theoretical analysis to derive the theoretical performance bound of the proposed CMED algorithm. Moreover, it is worth noting that although the system model and optimization problem are described based on the assumption of only one type of task, it can be easily extended to the scenario of providing services with different types. In particular, we just need to update the queue model by creating multiple queue backlogs for all service types, and then the replica number for each type of service can be determined after the simple algorithm extension.

C. THEORETICAL ANALYSIS AND DISCUSSION

According to the Lyapunov stability theorem, the value of the drift-plus-penalty function of the system should be within a threshold range for keeping the system in a stable state. Assuming the threshold value is $K$ when the system is stable, the following expression can be obtained according to the Lyapunov optimization theory:

$$
K = B + V_1 \cdot c_p^* + V_2 \cdot c_o^* - \epsilon \cdot Q(t),
$$
where $B$, $V_1$, $V_2$, and $\epsilon$ are the preset parameters. Assume that $c^e_p$ and $c^e_o$ are the average system cost and the average cost of replica deployment when the system achieves the target state, which means that for any time slot at this point, $\mathbb{E}[c_p(t)Q(t)] = c^e_p$ and $\mathbb{E}[c_o(t)Q(t)] = c^e_o$ are true under the condition of a stable system. In addition, it is easy to prove that there is a real number $\epsilon > 0$, which makes the formula $\mathbb{E}[\lambda(t) - \mu(t)|Q(t)|] \leq -\epsilon$ true.

Then, the system reaches the expected stable state when the value of the drift-plus-penalty function $f(t)$ is less than the threshold value $K$; that is, the following formula should be satisfied:

$$f(t)|Q(t)| \leq K. \quad (23)$$

Simultaneously, (14), (22) and (23), we have:

$$\mathbb{E}[\Delta(t) + V_1 \cdot c_p(t) + V_2 \cdot c_o(t)] \leq B + V_1 \cdot c^e_p + V_2 \cdot c^e_o - \epsilon \cdot Q(t). \quad (24)$$

By taking the expectation on both sides of (23) at the same time, we convert the conditional expectation into the expectation expression. Thereby, we have the following expression:

$$\mathbb{E}[\Delta(t) + V_1 \cdot c_p(t) + V_2 \cdot c_o(t)] \leq B + V_1 \cdot c^e_p + V_2 \cdot c^e_o - \epsilon \cdot \mathbb{E}[Q(t)]. \quad (25)$$

Substituting (13) into (25), we have:

$$\mathbb{E}[L(t) + 1 - L(t)] + \mathbb{E}[V_1 \cdot c_p(t) + V_2 \cdot c_o(t)] \leq B + V_1 \cdot c^e_p + V_2 \cdot c^e_o - \epsilon \cdot \mathbb{E}[Q(t)]. \quad (26)$$

By accumulating (26) over time slots $i \in [0, 1, \ldots, t - 1]$, the following relationship can be obtained:

$$\mathbb{E}[L(t)] - \mathbb{E}[L(0)] + V_1 \sum_{i=0}^{t-1} \mathbb{E}[c_p(i)] + V_2 \sum_{i=0}^{t-1} \mathbb{E}[c_o(i)] \leq B \cdot t + V_1 \cdot c^e_p \cdot t + V_2 \cdot c^e_o \cdot t - \epsilon \cdot \sum_{i=1}^{t-1} \mathbb{E}[Q(t)]. \quad (27)$$

According to (13), it can be known that for any time slot $t$, $L(t) \geq 0$ exists. In addition, according to (4), (8), and (9), it can be known that for any time slot $t$, $c_p(t) \geq 0$, $c_o(t) \geq 0$ exist. For the queue backlog $Q(t)$, the following equation can be obtained through the transformation of (27):

$$\frac{1}{t} \sum_{i=0}^{t-1} \mathbb{E}[Q(i)] \leq \frac{B + V_1 \cdot c^e_p + V_2 \cdot c^e_o}{\epsilon} + \frac{\mathbb{E}[L(0)]}{\epsilon \cdot t}. \quad (28)$$

From the relationship in (28), we can find that the practical limitation of queue backlog $Q(t)$ is restricted to some constant value. In particular, the upper bound of $Q(t)$ can be determined by the control coefficient $V$. The larger the value of $V$, the longer time average queue backlog will be created. Thus, the parameter $V$ can be chosen to control the upper bound of $Q(t)$ based on the buffer’s capacity in edge devices.

For the system cost, the following equation can be obtained through the transformation of (27):

$$\frac{1}{t} \sum_{i=0}^{t-1} \mathbb{E}[c_p(i)] \leq \frac{B}{V_1} + \frac{c^e_p}{\omega_1} + \frac{\mathbb{E}[L(0)]}{V_1 \cdot t}. \quad (29)$$

$$\frac{1}{t} \sum_{i=0}^{t-1} \mathbb{E}[c_o(i)] \leq \frac{B}{V_2} + \frac{c^e_o}{\omega_2} + \frac{\mathbb{E}[L(0)]}{V_2 \cdot t}. \quad (30)$$

$$\frac{1}{t} \sum_{i=0}^{t-1} \mathbb{E}[C(i)] \leq \frac{B}{V} + \frac{\omega_2 \cdot c^e_o}{\omega_1} + \frac{\omega_1 \cdot c^e_p}{\omega_2} + \frac{\mathbb{E}[L(0)]}{V \cdot t}. \quad (31)$$

When the system runs long enough, which means $t \to \infty$, it can be found from (29), (30), and (31) that $V_1$ and $V_2$ will increase correspondingly as the coefficient $V$ increases, which will lead to the reduction of system cost and the increasing of the task queue backlogs. Therefore, the coefficient $V$ can be used to make a trade-off between the system cost and the number of backlog tasks in the queue.

V. PERFORMANCE EVALUATION

In this section, extensive experiments have been conducted to evaluate the performance of our proposed algorithm. In the following, we first present the preliminary experiments to generate the practical data trace and system parameters used in the simulation. Then, we will present the evaluation setup and the corresponding results.

A. PRELIMINARY EXPERIMENTS

We consider deploying the Kubernetes cluster on the physical device, which contains a Master node (Ubuntu 18.04/12GB/8 Core) and three Worker nodes (Ubuntu 18.04/4GB/2 Core) with the same physical configuration and use the customized ResNet-50 model-based visual recognition application for task processing. Then, ImageNet is used as the data set to conduct the pre-experimental measurement, and we finally obtain the following basic parameters: for every single replica, the time required from the initialization to the successful service delivery is about 10ms, the average time required to complete a task is about 20ms, and the additional memory required is about 138Mb.
The simulation experiments will be carried out based on these parameters.

In order to simulate the dynamic task arrival and the different processing performance when the system is faced with individual tasks, the service rate of each replica and the task arrival in each time slot need to be randomized according to a certain distribution. In our simulation, workload traces in our experiments are generated based on the Poisson distribution model with a constant request arrival rate. The assumption of Poisson distribution-based job arrival is reasonable and has been demonstrated by the measurement study [22].

For the service rate of each Pod replica, we study the distribution of the service rate per time slot of each Pod through the preliminary experiments. In particular, we processed the 100,000 data samples in ImageNet through ResNet-50 model in the physical environment mentioned above. As shown in Figure 4, the experimental results show that the service rate distribution per time slot of each Pod obeys the exponential normal distribution approximately. Thus, the exponential normal distribution is adopted to generate the service rate of each Pod replica randomly.

**B. SIMULATION SETUP**

In our simulation, we set the duration of each time slot $t$ to 100 ms, and each simulation is conducted over 10,000 time slots. Limited by the physical resource, the maximum number of Pod replicas allowed by system $n_{\text{max}}$ is set to 10. For the physical cost coefficient $\omega_1$ and the operation cost coefficient $\omega_2$ in CMED, we simply set the ratio of $\omega_2$ to $\omega_1$ (i.e., $\frac{\omega_2}{\omega_1}$) to 1.

We compare the performance of CMED with the following three types of service deployment strategies:

- **RANDOM.** A random value is generated as the number of Pod replicas per time slot. The distribution of the number of Pod replicas is decided by the average system workload $\lambda_{\text{avg}}$ and the average service rate of each Pod $C_{\text{rate}}$, which is obtained through the preliminary experiments. Here, the number of Pod replicas $n(t)$ in the RANDOM obeys the normal distribution with $N(\frac{\lambda_{\text{avg}}}{C_{\text{rate}}}, \frac{\lambda_{\text{avg}}}{C_{\text{rate}}^2})$.

- **CONSTANT.** A constant value is adopted as the number of Pod replicas per time slot. This strategy selects the median value between the minimum number and the average number as the constant number of Pod replicas.

- **ARIMA.** The study in [15] proposed a microservice elastic deployment scheme by dynamically adjusting the number of Pod replicas according to the predicted workload, which is achieved based on the model ARIMA. Here, we establish the ARIMA model in advance based on the historical workload information to predict the future workload in a certain window, and updated the model with the real-time workload to ensure the reliability of the forecast.

Two experimental scenarios are conducted to investigate the performance of CMED: (1) Sensitivity to the parameter $V$. In this subset of experiments, we study the sensitivity performance of CMED by changing the value of control parameter $V$. In particular, the average workload $\lambda$ is fixed to 20 and the control parameter $V$ varies from $10^{-3}$ to $10^5$. (2) Sensitivity to the average workload. In this subset of experiments, we study the performance of CMED under different workloads. In particular, the control parameter $V$ is fixed to $10^3$ and the average workload $\lambda$ varies from 0 (not included) to 80.

To evaluate the performance of CMED with respect to the system cost, delay, and the tradeoff, we collected the average system cost, queue backlogs, and the number of Pod replicas per time slot as the evaluation metrics. The data points in the following results are averaged over 10,000 runs.

**C. EXPERIMENT RESULTS**

1) **SENSITIVITY TO PARAMETER V**

Figure 5 illustrates the performance versus different values of $V$. Since the performance of RANDOM, CONSTANT and ARIMA are not affected by the control coefficient $V$, the corresponding curve will retain the same value as $V$ increases (RANDOM curve will fluctuate slightly due to the randomness). As expected, we can see that the value of task queue backlogs stays on a high level for CONSTANT mostly, because its deployment policy keeps the number of Pod replicas at a fixed level and the system cannot provide enough services in most cases. On the contrary, RANDOM achieves lower task queue backlog which is close to zero, because we use the normal distribution of the number of Pod replicas of the historical workload to ensure the system QoS, which means the system can meet the requirement of workload in most cases. For ARIMA, we can observe from the figure that the number of Pod replicas deployed by the system decision is about 4, because the average arrival rate of tasks in the experiment is set to 20, and the average processing rate of tasks with a single replica is set to 5. However, this does not necessarily meet system requirements, because the predicted values are used for the decision on deployment and the predicted results may not match the actual arrival of the task. As a result, the system always has queue backlogs under ARIMA. For CMED, it’s obvious that the system cost is always below 2.5 and the average value of queue backlogs is zero, which are both lower than the values in RANDOM when $V < 10^4$. In fact, the system’s average queue backlog
is a reflection of the QoS of the system. In general, with the increase of $V$, the number of Pod replicas under CMED will decrease, resulting in a reduction in system cost and QoS.

2) SENSITIVITY TO WORKLOAD
Figure 6 illustrates the performance versus different workload. As expected, we can observe from Figure 6 that the system cost and queue backlogs of all approaches but not CONSTANT increase as the workload $\lambda$ increases, because the number of Pod replicas of CONSTANT is fixed. In addition, it can be observed that the cost and queue backlog of CMED is lower than RANDOM when the workload is at a moderate level ($15 < \lambda < 30$). At a lower level of workload ($\lambda < 15$), it’s allowed to have a queue backlog to achieve lower system cost in RANDOM because of the low weight of QoS. When the workload is not high ($\lambda < 40$), ARIMA shows a lower system cost than RANDOM and CMED, but there are different levels of queue backlogs at the same time. In addition to the reason that the predicted value does not accurately match the real-time situation of the system, the lower the workload, the easier it is to generate more NaN values when the data is differentially processed, which leads to a decrease in the prediction accuracy. Overall, CMED can be said to be able to effectively achieve performance goals in terms of system cost and QoS within a certain workload. However, it can be found that the number of queue backlogs in CMED is no longer regulated when the workload $\lambda > 55$. This is because the physical resource in the system cannot support more replica deployment, and the number of deployed replicas has reached the maximum limit of the system.

VI. FINAL REMARK
In this paper, we studied the issue of how to elastically deploy the container-based microservices with a cost-efficient way for the edge computing service system with unpredictable knowledge. In particular, we developed a cost-aware elastic microservice deployment algorithm, which is capable of minimizing the system cost caused by the microservice deployment, while achieving the QoS guarantee and system stability. Based on the Lyapunov optimization framework, the proposed algorithm makes the microservice deployment...
Towards Cost-Efficient Edge Intelligent Computing With Elastic Deployment

P. Zhao et al.

Plan greedily without the future knowledge of the system and users. Simulation results demonstrate that CMED outperforms the baseline schemes, and provides an efficient tradeoff between user’s QoS and system cost. As ongoing research, an extension work is being conducted to study the issues of dynamic service deployment and service placement for the system with multiple service types in geographically distributed heterogeneous edge devices.

REFERENCES

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” IEEE Internet Things J., vol. 3, no. 5, pp. 637–646, Oct. 2016.

[2] M. Satyanarayanan, “The emergence of edge computing,” Computer, vol. 50, no. 1, pp. 30–39, Jan. 2017.

[3] H. Li, K. Ota, and M. Dong, “Learning IoT in edge: Deep learning for the Internet of Things with edge computing,” IEEE Netw., vol. 32, no. 1, pp. 96–101, Jan. 2018.

[4] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, “Edge intelligence: Paving the last mile of artificial intelligence with edge computing,” Proc. IEEE, vol. 107, no. 8, pp. 1738–1762, Aug. 2019.

[5] X. Wang, Y. Han, V. C. M. Leung, D. Niyato, X. Yan, and X. Chen, “Convergence of edge computing and deep learning: A comprehensive survey,” IEEE Commun. Surveys Tuts., vol. 22, no. 2, pp. 869–904, 2nd Quart., 2020.

[6] E. Li, L. Zeng, Z. Zhou, and X. Chen, “Edge AI: On-demand accelerating deep neural network inference via edge computing,” IEEE Trans. Wireless Commun., vol. 19, no. 1, pp. 447–457, Jan. 2020.

[7] Gartner. Edge Computing Promises Near Real-Time Insights and Facilitates localized Actions. Accessed: Mar. 2020. [Online]. Available: https://www.gartner.com/smarterwithgartner/what-edge-computing-means-for-infrastructure-and-operations-leaders/

[8] MarketsandMarkets. Edge Computing Market by Component (Hardware, Platform, and Services), Application (Smart Cities, IoT, Content Delivery, Remote Monitoring, AR and VR), Organization Size (SMES and Large Enterprises), Vertical, and Region-Global Forecast to 2024. Accessed: Mar. 2020. [Online]. Available: https://www.marketsandmarkets.com/Market-Reports/edge-computing-market-133384090.html

[9] P. Sharma, L. Chaufournier, P. Shenoy, and Y. C. Tay, “Containers and virtual machines at scale: A comparative study,” in Proc. 17th Int. Middleware Conf., Nov. 2016, pp. 1–13.

[10] K. Dolui and C. Kiraly, “Towards multi-container deployment on IoT gateways,” in Proc. IEEE Global Commun. Conf. (GLOBECOM), Dec. 2018, pp. 1–7.

[11] A. Buzachis, A. Galletta, L. Carnevale, A. Celesti, M. Fazio, and M. Villari, “Towards osmotic computing: Analyzing overlay network solutions to optimize the deployment of container-based microservices in fog, edge and IoT environments,” in Proc. IEEE 2nd Int. Conf. Fog Edge Comput. (ICFEC), May 2018, pp. 1–10.

[12] X. Wan, X. Guan, T. Wang, G. Bai, and B.-Y. Choi, “Application deployment using microservice and docker containers: Framework and optimization,” J. Netw. Comput. Appl., vol. 119, pp. 97–109, Oct. 2018.

[13] P. Singh, P. Gupta, K. Jyoti, and A. Nayyar, “Research on auto-scaling of Web applications in cloud: Survey, trends and future directions,” Scalable Comput., Pract. Exper., vol. 20, no. 2, pp. 399–432, May 2019.

[14] M. Alam, J. Rufino, J. Ferreira, S. H. Ahmed, N. Shah, and Y. Chen, “Orchestration of microservices for IoT using docker and edge computing,” IEEE Commun. Mag., vol. 56, no. 9, pp. 118–123, Sep. 2018.

[15] R. N. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya, “Workload prediction using ARIMA model and its impact on cloud applications’ QoS,” IEEE Trans. Cloud Comput., vol. 3, no. 4, pp. 449–458, Oct. 2015.

[16] F. H. L. Buzato, A. Goldman, and D. Batista, “Efficient resources utilization by different microservices deployment models,” in Proc. IEEE 17th Int. Symp. Netw. Comput. Appl. (NCA), Nov. 2018, pp. 1–4.

[17] A. Bhattacharjee, A. D. Chhokra, Z. Kang, H. Sun, A. Gokhale, and G. Karsai, “BARISTA: Efficient and scalable serverless system for deep learning prediction services,” in Proc. IEEE Int. Conf. Cloud Eng. (ICE), Jun. 2019, pp. 23–33.

[18] M. Lin, A. Wierman, L. L. Andrew, and E. Theresa, “Dynamic right-sizing for power-proportional data centers,” IEEE/ACM Trans. Netw., vol. 21, no. 5, pp. 1378–1391, Oct. 2013.