Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH

Open Access

SLA-ORECS: an SLA-oriented framework for reallocating resources in edge-cloud systems

Shizhan Lan[1,3], Zhuoxi Duan[2*], Song Lu[1], Bin Tan[1], Shi Chen[1], Yeyu Liang[1] and Shan Chen[1]

Abstract

The emergence of the Fifth Generation (5G) era has ushered in a new era of diverse business scenarios, primarily characterized by data-intensive and latency-sensitive applications. Edge computing technology integrates the information services environment with cloud computing capabilities at the edge of the network. However, the evolving landscape of business models necessitates a unified edge architecture capable of accommodating diverse requirements, posing substantial challenges for service providers in meeting Service-Level Agreements (SLAs).In response to these challenges, we introduce SLA-ORECS. This innovative framework dynamically allocates dedicated and shared resources within the edge-cloud system to cater to service requests with varying SLAs, thereby facilitating performance isolation. Furthermore, we have developed an optimization algorithm to enhance the efficiency of SLA assurance during request dispatch.The evaluation of SLA-ORECS highlights its noteworthy performance improvements, particularly in terms of system throughput and average time consumption.

Keywords Edge-cloud system, Service-level agreement (SLA), Performance isolation

Introduction

With the material abundance of life, mankind has become more focused on the pursuit of smart living, with more and more smart devices flooding into everyday life. This trend has led to diverse and urgent requests constantly being generated at the edge of the network, placing even greater demands on the already congested backbone and mobile networks. At present, edge computing technology is emerging to diffuse cloud computing capabilities to edge clusters equipped with computation/storage capacity, allowing service requests to be processed closer to the data sources. This not only relieves the pressure on the backbone network but also provides more favorable service quality assurance (e.g., privacy protection, timely response) for service requests.

Some pioneer work provide solutions for optimization problems within a heterogeneous edge-cloud system under various settings and demonstrate significant performance through simulations. For example, Li et al. [1] investigate the multi-user dual computation offloading via a hybrid non-orthogonal multiple access and frequency division multiple access transmission. With the objective of minimizing the overall latency for completing all wireless devices' tasks, they also propose a layered yet cell-based distributed algorithm for finding the optimal dual offloading solution. In order to minimize the average application response time, Guo et al. [2] focus on the problem of how to offload computationally intensive applications and assign the bandwidth and propose an efficient algorithm to find the optimal solution. In addition, Kamran et al. [3] propose a framework for joint computation scheduling, caching, and request forwarding within decentralized computing environments and develop a throughput optimal control policy.

*Correspondence:
Zhuoxi Duan
Daunzhuoxi666@163.com
[1] China Mobile Guangxi branch Co., Ltd, Nanning 530012, China
[2] School of Computer Science and Technology, Tianjin University, Tianjin 300354, China
[3] School of Software Engineering, South China University of Technology, Guangzhou 530012, China

The emergence of new technologies like 5G has introduced possibilities to enhance the quality of existing services and introduce new ones, such as enhanced mobile broadband, IoT applications, augmented reality, critical mission services, cloud gaming, and intelligent infrastructure. Despite the increased responsiveness to various requests, challenges such as network congestion, bandwidth limitations, scalability issues, service differentiation, or security concerns persist, making it challenging to accommodate additional users and services without disrupting normal operations. Service Level Agreements (SLAs) [4] represent commitments made by service providers to customers regarding service parameters. For instance, Internet service providers will reach SLAs with customers, which usually have a technical definition in mean time between failures, throughput, jitter, or similar measurable details. While these previous works [1–3] show excellent simulation performance in their settings, none of them focus on SLA-related issues: *1) Different performance requirements between different services.* In reality, services in different scenarios have different SLAs. For example, autonomous driving requires lower latency, while video transmission requires higher bandwidth. *2) Service performance guarantee.* Only the service performance guarantee can bring better experience to users.

The Fifth Generation (5G) is expected to be able to meet the different SLAs of users. Based on Network Functions Virtualization (NFV) and Software-Defined Networking (SDN), network slicing is proposed as a new paradigm for building service-customized 5G networks. Some studies have attempted to address SLA-related issues through resource allocation or resource orchestration [5–10], with the aim of improving revenue or enhancing system performance. Liu et al.. design a constraint-aware deep reinforcement learning algorithm that learns the resource orchestration policy to allow optimization of network slicing dynamically without violating the SLAs of slices [8]. Alsaffar et al. present a new strategy for optimizing big data distribution and propose an algorithm to allocate resources to meet SLA and Quality of Service (QoS) requirements [10]. However, there is still a lack of systematic discussion of "dynamic SLA customization with SLA guarantee" in the edge-cloud system.

The Deep Reinforcement Learning (DRL) algorithm has demonstrated efficient decision-making capabilities in resource management problems and has seen widespread application. Liu et al. [11] introduced a DQN-based algorithm to maximize the long-term computational resource utility in MEC networks. Zhao et al. [12] employed Dueling Double DQN for efficient user association in cellular networks. In another study, [13] proposed an asynchronous advantage actor-critic scheme for adaptive bandwidth allocation in wireless access networks, ensuring reliable transmission of video data streams. Zhang et al. [14] utilized a Deep Weighted Double Q-learning algorithm to learn dynamic caching policies in ultra-dense networks, minimizing latency. Furthermore, in [15], the Deep Deterministic Policy Gradient (DDPG) algorithm was employed to address joint resource optimization problems in vehicular networks. These DRL-based solutions significantly enhance resource utilization in wireless networks. However, neural networks excel in learning but not in search, whereas conventional search algorithms are adept at search but not learning. Traditional optimization algorithms tend to converge faster and exhibit higher stability compared to DRL. To overcome these limitations, our work combines artificial intelligence with optimization algorithms, employing DRL for resource customization while utilizing optimization algorithms for service coordination.

Therefore, this paper investigates the SLA-oriented resource allocation and a request dispatch problem within the edge-cloud system to improve system performance by implementing dynamic SLA customization and SLA guarantees. Through SLA customization, resources are allocated among requests with different SLAs so that requests with the same SLA can be dispatched within an identical ***channel***, which we call performance isolation (PI). At the same time, we study the in-channel scheduling problem to provide better user experiences with guaranteed SLAs.

A dynamic performance isolation effectively avoids vicious resource competition and thus increases the number of requests that satisfy the SLA, i.e., the throughput rate, due to the fact that there is resource competition between requests with different SLAs. Imagine a scenario where a service consumes most of the memory resources on a node. In this case, the scheduling of other services will be severely adversely affected, even if they require only a small amount of memory resources. In addition, requests within the same channel have the same SLA, which makes it easier to model optimization problems that take SLA guarantees into account. In addition, under performance isolation, request scheduling and service orchestration are channel-oriented; thus, only specific services within the channel participate in the computation. Since the channels are independent, they can run in parallel, thus saving a lot of time. To the best of our knowledge, this work is the first to study resource management in edge-cloud systems under the dual conditions of dynamic performance isolation and SLA guarantees. Our contributions can be summarized as follows.

- We propose a Resource Redefined Edge-Cloud Collaborative Framework for Service-Level Agreements,

SLA-ORECS, as shown in Fig. 1, which aims to dynamically achieve resource allocation among channels with different SLAs through a two-step process of resource customization and service orchestration. Resource customization refers to dividing dedicated resources among different channels, while service orchestration depends on the binding of shared and dedicated resources.

- We adopt a two-time-scale mechanism to present an implementation of the proposed framework, as illustrated in Fig. 1. It combines artificial intelligence and optimization algorithms, using Deep Reinforcement Learning (DRL) for resource customization and optimization algorithms for service orchestration.
- We provide a proof-of-concept evaluation and validate the advantages of *SLA-ORECS* in terms of system performance. It shows promising performance in system throughput and average time consumption.

Illustration of *SLA-ORECS* framework

This section describes *SLA-ORECS* and introduces its two major steps: resource customization and service orchestration.

System overview

Edge computing refers to the provision of low-latency computing services at or near the physical location of users or data sources by leveraging the storage and computation capabilities of the edge infrastructure near the end devices. As shown in Fig. 1, we design *SLA-ORECS* based on the traditional edge-cloud system architecture, which consists of three layers: end-device layer, edge node layer, and cloud layer.

The end device layer is composed of various user devices that generate real-time service requests, such as cameras, mobile phones, etc., which are usually connected to edge clusters through wireless communications. For example, a camera captures real-time media data for object detection.

The edge node layer comprises numerous distributed facilities/infrastructures, such as local servers or base stations, responsible for receiving requests from end devices and making decisions about the destination of request scheduling based on conditions such as resource distribution. As a beneficial supplement to the cloud layer, the edge node layer is more suitable for local processing of small-scale delay-sensitive computing tasks. If resources permit, the edge node layer can independently process requests without transmitting them to the cloud through the backbone network.

The cloud layer is generally composed of server clusters with massive computation and storage capacities, responsible for complex data processing and large-scale data/service storage. It can cooperate with the edge nodes to take over service requests that are not delay-sensitive but resource-intensive. The most common form of cloud layer is commercial cloud platforms, such as Alibaba Cloud or Amazon EC2.

Requirements for *SLA-ORECS*

With high bandwidth, high reliability, low latency, and large-scale interconnection, 5G technology will give

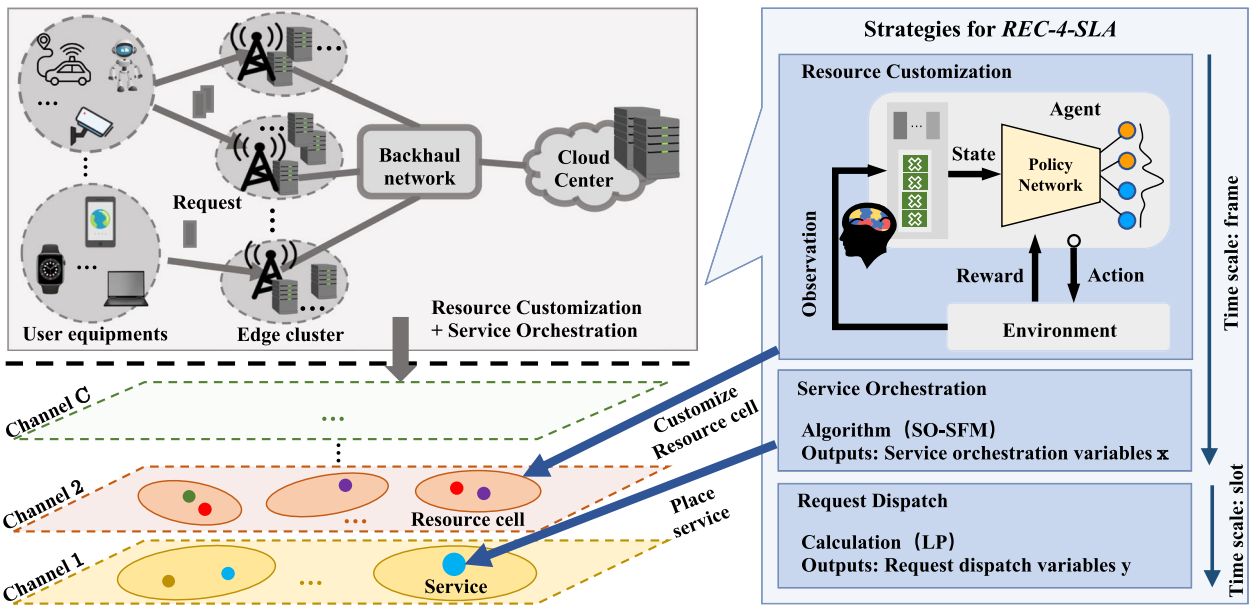

Fig. 1 Framework to support dynamic performance isolation in the edge-cloud system

consumers more emerging services and more diversified experiences. Therefore, the service scenarios under the edge-cloud collaboration will be more diverse in the QoS requirements and key performance indicators. For example, an edge-cloud system may simultaneously require high-quality video streaming, machine-type communications, Internet of Things (IoTs) with ultra-low-rate communications from a large number of devices, and automotive and haptic applications with millisecond latency. At the same time, with the further expansion of the network scale and the increasing demand for high-quality network services, the current network is already facing severe tests. These trends drive the construct of a dynamically tailored edge-cloud system to support services with specific SLAs and the design of intelligent policies to guarantee SLAs.

Steps to implement *SLA-ORECS*

Typically, processing data-intensive service requests (e.g., augmented reality, video analytics, distributed machine learning) requires not only a dedicated share of resources (CPU cycles, memory for computation, storage resources for storage) but also a large amount of data on the server (e.g., object databases, trained machine learning models). Therefore, we divide the core process of *SLA-ORECS* into two steps: 1) resource customization and 2) service orchestration, where the first step is the allocation of dedicated resources and the second step is the binding of service replicas to dedicated resources. With the two steps, the resources of edge nodes and the cloud are cut into many mutually isolated combinations of dedicated and shared resources, each of which we call a resource cell or ***cell*** for short. In addition, the set of cells corresponding to an SLA is called a channel.

Resource customization

Resource customization implies allocating dedicated resources in advance of request dispatch, not only between channels but also between cells. We classify cells into two types based on the combination of resources within a cell: horizontal and vertical cells. A ***horizontal cell*** means that resources of the cell are all from one edge node or several edge nodes. For example, some resources from edge node n_1 and some from edge node n_2 together form a joint cell. This horizontal cell means that the cell's subject judges that more responsive resources from edge nodes are needed for the current situation rather than massive cloud resources. A ***vertical cell*** indicates that resources of the cell are from the edge node layer partly from the cloud layer. For example, a cell's subject builds a cell consisting of the union of cloud resources and the resources of edge node n_1. This vertical cell probably indicates most current requests are characterized by low latency sensitivity but high resource requirements, so cloud resources are needed more than edge node resources.

Service orchestration

With the first step, resource customization, each channel is pre-allocated with some cells containing merely dedicated resources. The next step is to consider the orchestration of shared data, and what needs to be done is to place a wide range of service replicas (shared code, well-trained machine learning models, etc.) with specific SLA on cells of the corresponding channel. For each selected single orchestration, e.g., a replica of service l is placed on cell m, which we denote as a binary group (l, m). Then we can define S to be the set of selected single service orchestrations. Thus the service orchestration problem can be formulated as a set optimization problem.

Since service orchestration is closely related to subsequent request dispatch, we consider it jointly and model it as an optimization problem, which will be introduced and solved in the next section.

Formulation

This section first introduces the system model, then describes the *SLA-ORECS* designed to achieve dynamic performance isolation and SLA guarantee. To allocate the two types of resources required by data-intensive applications, we design a solution that first implements resource customization with DRL. According to our proposed algorithm, an approximately optimal service orchestration solution is generated within each channel. In this way, performance isolation is achieved in the edge-cloud system by combining an artificial intelligence algorithm and an optimization algorithm. Based on the allocation of dedicated and shared data results, the in-channel request scheduling problem to maximize throughput (the number of requests served with guaranteed SLAs) becomes a Linear Programming (LP) problem.

System model

As illustrated in Fig. 1, there is an edge-cloud system providing coverage to a generic geographic region, where a group of edge clusters covers a set of end devices. We first consider a basic edge-cloud system that contains an edge cluster and the cloud data center, with an edge cluster consisting of a set of edge nodes $\mathcal{N} = \{n_1, ..., n_N\}$. Each edge node has limited storage, computation, and other dedicated resources, and the capabilities of different edge nodes can be heterogeneous. The cloud data center has relatively high capacities compared to edge nodes.

There is a set $\mathcal{C}$ of SLAs, one of which $c \in \mathcal{C}$ consists of a service subset $\mathcal{L}_c$. Services are heterogeneous from one another, not only in terms of resources but also in SLAs

such as the maximum response time (the lifecycle of service). The existence of each SLA implies the need for a channel, so the set of channels is also $\mathcal{C}$. We use DRL techniques for resource customization and an optimization algorithm for service orchestration, in which each channel $c \in \mathcal{C}$ is allocated a set of heterogeneous cells $\mathcal{M}_c$ containing resources such as computation, service replicas.

A channel containing fixed cells would have poor flexibility, so a dynamic *SLA-ORECS* is better than a static *SLA-ORECS*. In addition, considering cost reduction and stability, we adopt a two-time-scale mechanism depicted by Fig. 1 [16] to schedule request dispatch, resource customization, and service orchestration. *SLA-ORECS* performs request dispatch at a smaller scale, slot t while carrying out resource customization and service orchestration at a larger scale, frame f.

Resource customization

Resource customization by DRL

DRL, a combination of Reinforcement Learning (RL) and Deep Learning (DL), has opened up many new applications in finance, robotics, healthcare, smart grid, and other areas. Advanced research results in this field have been able to solve more complex decision-making tasks that were previously beyond the reach of machines.

Resource customization is a particularly tedious process, with the generation of each cell involving varied channels, multiple nodes (edge nodes or the cloud center), and various resources selection decisions. If solved using traditional heuristic algorithms, it suffers from overly complex modeling and high solution complexity. Therefore, we customize the DRL algorithm for resource management in edge-cloud systems, i.e., using DRL for resource customization. Because it does not require to perform complex modeling but only key factors as state space inputs, and the DRL model can continuously learn by exploring and receiving feedback from the environment to continuously optimize the utility.

However, one key challenge is still pending in practice, namely the deployment of DRL agents. In this case, we design the resource customization mechanism under single-agent decisions, i.e., deploying a DRL agent on each edge node responsible for the decision-making of cells dominated by that edge node.

For rationally managing resources in edge-cloud systems with numerous SLAs, the agent should make decisions about allocating multiple dedicated resources in terms of a control action (c, a, r), where $c \in \mathcal{C}$ is the channel selection decision, denoting which SLA the cell serves, a specifies the edge nodes or cloud that jointly make up the cell, and r is the resource contribution profile, indicating how much each selected server contributes to various resources of the cell.

The whole problem can be summarized as follows: according to a fixed control policy $\mathbf{\Phi} = (\Phi_c(\mathbf{\Theta}), \Phi_a(\mathbf{\Theta}), \Phi_r(\mathbf{\Theta}))$, the edge node decides which channel the cell belongs to, the selection of joint nodes (edge nodes or the cloud center) and the resource composition of the cell. At the same time, it keeps observing the current network state $\mathbf{\Theta}$, including the request rate of each service and the state of all cells in each channel. It is worth noting that even for the same frame the network state perceived by nodes is different if there is an order of edge nodes building cells. Because the previous cells change the current environment, the later edge node can perceive cells of the previous one. In addition, we define a utility function $\psi(\mathbf{\Theta}, (c, a, r))$ as reward function to evaluate the overall effectiveness of the system, which is proportional to the overall throughput rate of the system. By taking advantage of DRL algorithms, such as Proximal Policy Optimization (PPO) [17], the control policy $\mathbf{\Phi} = (\Phi_c(\mathbf{\Theta}), \Phi_a(\mathbf{\Theta}), \Phi_r(\mathbf{\Theta}))$ can be trained and achieve increased utility of the system for long-term performance.

A case of resource customization

As shown in Fig. 2, we offer an example of the entire process of resource customization. For illustrative purposes, it is assumed that multiple rounds of resource customization are carried out in numbered order. First, the edge node n_1 takes the observed state $\mathbf{\Theta}_1$ as input and outputs decisions based on the control policy $\mathbf{\Phi}$. The decision process is as follows: this cell belongs to the channel 2, the assisting nodes for the joint cell are edge node n_3 and the cloud, and the cell's composition of dedicated resources is given. After edge node n_1 finishes this resource customization, it makes a change to the current environment, so when edge node n_2 observes the environment, the state has been changed to $\mathbf{\Theta}_2$, and a decision is made accordingly. After multiple rounds of resource customization, the resources of edge nodes and the cloud center are allocated to cells of various channels.

Service orchestration

In this subsection, we use the channel $c \in \mathcal{C}$ as an example. We formulate the problem of service orchestration and request dispatch to maximize throughput. Moreover, a mechanism is designed to solve the two sub-problems using the same method. At the same time, we present an algorithm based on submodular function maximization [18] to select the service orchestration solution.

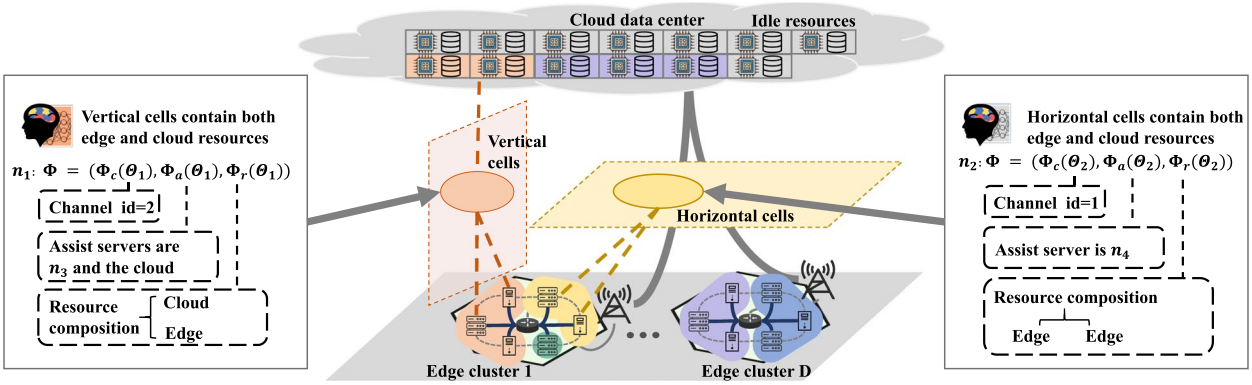

Fig. 2 A case illustration of resource customization in the edge-cloud system

Task model

To maximize the throughput in the channel, we set two decision variables $\mathbf{x}$ and $\mathbf{y}$ to model the mathematical problem, where $\mathbf{x}$ are the service orchestration variables and $\mathbf{y}$ are the request dispatch variables. $x_{lm} \in \{0, 1\}$ is 1 if the replica of service l is placed on cell m and 0 otherwise. $y_{lnm} \in [0, 1]$ represents the probability that a request of service l submitted to edge node n is dispatched to cell m. As the channels are independent of each other, it can simultaneously maximize the edge-cloud system's throughput.

$$\begin{aligned} P1: &\ \max \boldsymbol{Channel\ Throughput}\ (\mathbf{x}, \mathbf{y}) \\ \mathbf{s.t.}\ &C_u^1: \mathit{ReplicaStorage} \le \mathit{Storage}, \text{foreachcell} \\ &C_u^2: \mathit{RequestResource} \le \mathit{Resource}, \text{foreachcell} \\ &C_u^3: \mathbf{y} \le \mathbf{x} \cdot \mathbb{1}_{SLA} \\ &C_u^4: \mathbf{x}, \mathbf{y} \in \mathit{ValidRange} \end{aligned} \tag{1}$$

The underlying optimization problem of service orchestration and request dispatch can be formulated as (1): the objective is to maximize the expected number of requests whose SLA is fully guaranteed per slot. Constraint C_u^1 ensures that each cell does not store more than its storage capacity, where the cell stores the sum of storage size of each service replica placed on that cell. Constraint C_u^2 guarantees that each cell is not dispatched with more requests than its resource capacity allows. Constraint C_u^3 ensures that a request is successfully dispatched if it enables guaranteed SLA, and the cell contains the requested service replica, where $\mathbb{1}_{SLA}$ is the indicator function. If the SLA of a request can be fully guaranteed, $\mathbb{1}_{SLA}$ is 1, otherwise 0. Through this constraint, the SLAs of requests are strictly guaranteed. Constraint C_u^4 specifies that decision variables take values within valid ranges.

Since the objective function and constraints of (1), i.e., ***Channel Throughput*** **(x, y)**, contains both integer variables $\mathbf{x}$ and real variables $\mathbf{y}$, it is a mixed integer programming problem, which is NP-hard and cannot be solved in polynomial time. Therefore, we decompose the problem into two sub-problems of request dispatch and service orchestration.

Solution for request dispatch

To reduce costs and improve stability, we separate the time scale of request dispatch (performed per slot) from resource customization and service orchestration (performed per frame). At each slot, service orchestration is already selected at the beginning of the frame, i.e., the decision variables $\mathbf{x}$ are known. Therefore, we can solve the sub-problem of (1) regarding $\mathbf{y}$ (see (2)), and perform probabilistic dispatch, where a request for service l submitted to edge node n will be dispatched to cell m with probability y_{lnm}. Since the decision of problem (2) is only related to the real variables $\mathbf{y}$, it is a linear programming [16] that can be solved in polynomial time.

$$\begin{aligned} P2: &\ \max \boldsymbol{Channel\ Throughput}\ (\mathbf{y}) \\ \mathbf{s.t.}\ &C_p^1: \mathit{RequestResource} \le \mathit{Resource}, \text{for each cell} \\ &C_p^2: \mathbf{y} \le \mu \cdot \mathbb{1}_{SLA} \\ &C_p^3: \mathbf{y} \in \mathit{ValidRange} \end{aligned} \tag{2}$$

where the constraints include resource finiteness C_p^1, SLA guarantee and service orchestration guarantee C_p^2, and range validity C_p^3. The binary variable μ represents whether the service replica is placed on the cell to which the request will be dispatched. As a result, request dispatch can strictly guarantee SLAs and meet resource requirements.

Algorithm for service orchestration

We reformulate the service orchestration problem as a set optimization. For each selected single orchestration,

e.g., a service l is placed on cell m, which we denote as a binary group (l, m). Then we can define $S \subseteq \mathcal{L}_c \times \mathcal{M}_c$ to be the set of selected single service orchestration. Let ***Channel Throughput*** **(*S*)**, which can be calculated by solving the request dispatch problem (see (2)), denote the optimal objective value of (1) under a fixed service orchestration set S or fixed decision variables $\mathbf{x}$, where $(l, m) \in S$ if and only if $x_{lm} = 1$. Then, we can rewrite the service orchestration problem as:

$$\begin{aligned} P3: \ & \max \textbf{\textit{Channel Throughput}} \ \textbf{(\textit{S})} \\ \textbf{s.t.} \ & C_h^1: \textit{ServiceStorage} \leq \textit{Storage}, \text{for each cell} \\ & C_h^2: S \subseteq \textit{DefinitionDomain} \end{aligned} \tag{3}$$

where the constraints include the finiteness of cells' storage capacity C_h^1 and the rationality of service orchestration set C_h^2.

Mechanism: In order to verify how beneficial a service orchestration case is to the current frame, we design an evaluation mechanism, as in Fig. 3. For a service orchestration case, we take the average predicted request rate of that frame to perform request dispatch. The resulting throughput in the current case is used as a criterion to evaluate this service orchestration. The average predicted request rate is the sum of expected request rates of all slots in the frame divided by the number of slots. The specific execution flow of the mechanism and the meaning of each step, as in Fig. 3, are shown below.

(a) Service orchestration variables and known parameters are involved as inputs to the problem (2).
(b) By solving the problem (2), the dispatch variables and the throughput rate can be derived.
(c) The throughput rate derived in step (b) is used as a criterion for judging the service orchestration set.
(d) Each service orchestration set iterated by the algorithm is converted to the corresponding service orchestration variables for participation in step (a).

Algorithm: Due to the submodularity of the objective function [18], we design a heuristic algorithm, Service Orchestration founded on Submodular Function Maximization (SO-SFM). In the process of SO-SFM, different service orchestration schemes are selected iteratively, and the approximately optimal one is established through the evaluation mechanism. To further illustrate, the detailed flow of the algorithm can be expounded by the following steps.

- Initialize the selected service orchestration set S as an empty set.
- Select the element u from the set $(\mathcal{L}_c \times \mathcal{M}_c) \setminus S$ that makes the set $S \cup \{u\}$ not only satisfy constraints of (3) but also maximize the objective function of (3).
- Merge the selected element u to the set S.
- Repeat the above two steps until there is no element in the set $(\mathcal{L}_c \times \mathcal{M}_c) \setminus S$ or S is already the most extensive set satisfying the constraints of (3).

Data-driven evaluation

Experimental settings

For investigating the performance of *SLA-ORECS*, simulations on request dispatch are presented for edge-cloud systems with different SLAs. Among all simulations, the time horizon is discretized into frames, where a frame includes 50 slots. We designed the experimental environment contains the cloud center and an edge cluster with

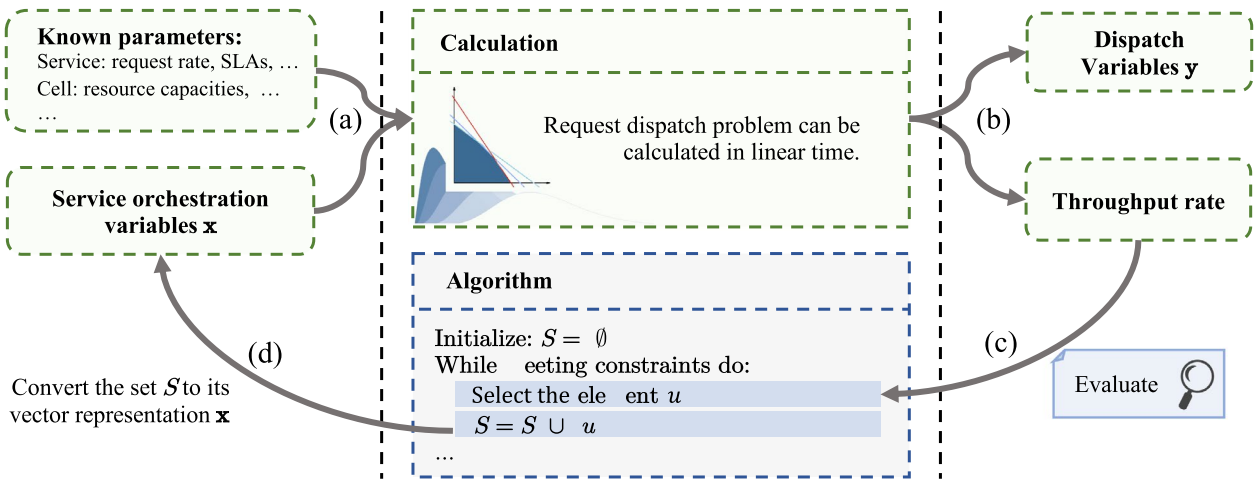

Fig. 3 Evaluation mechanism for service orchestration within a channel

four edge nodes. For illustrative purposes, we consider two kinds of dedicated resources, storage resources and computational resources and adopt maximum response time to indicate the difference of SLAs.

To study the role of performance isolation, we assume that an edge-cloud system contains $|\mathcal{C}| = 5$ SLAs, where there are $|\mathcal{L}_c| = 4$ or so services within each channel. To indicate the difference in SLAs, the requests in the 5 channels have different maximum response time. Services' range of data value and request distribution are based on the Alibaba Cluster Trace [19] to ensure *SLA-ORECS* has effective performance in the real environment. As for the DRL settings in edge nodes, We choose the PPO algorithm and select Relu as the activation function and Adam optimizer. For the settings of other parameters, we refer to the settings of this project [20].

Evaluation results

To collaborate the performance of *SLA-ORECS*, experiments on request dispatch are carried out under various settings. Fig. 4 gives the details of performance comparison as follows.

1) In Fig. 4(a), the performance of request dispatch under PI with PPO and SO-SFM is compared to No Performance Isolation (No-PI) with SO-SFM; We can see that when the DRL agent is trained to convergence, the request dispatch under PI is higher and more stable than that under No-PI in terms of throughput rate, which indicates the advantage of dynamic SLA customization.
2) We construct several baselines to show the performance advantage of the service orchestration with SO-SFM. In Fig. 4(b), the performance of SO-SFM is compared to other solutions: *probability orchestration*, *simulated annealing*, and *top-Q*. We divide the storage resource capacity of edge nodes into 6 levels (higher is better). It can be seen that with the increase of the storage resource capacity, the throughput rate shows an upward trend. In addition, the throughput rate of SO-SFM is always higher than other baselines, which also demonstrates the superiority of the optimization algorithm. *Note: top-Q* mentioned above, which sequentially considers each cell, computes the total demand for each service that can be dispatched, and then perform service orchestration in descending order until reaching storage limitation; *probability orchestration* means to firstly solve the LP relaxation of (1), and then place service replicas in descending order of the service orchestration variables (can be seen as orchestration probability) subject to the storage constraints; *simulated annealing* [21] is a stochastic optimal search strategy for iterative solutions, whose starting point is based on the similarity between the annealing process of substances in physics and general combinatorial optimization problems.
3) We also collect the computational time during request dispatch and service orchestration in an edge-cloud system with heterogeneous SLAs. Under PI, both request dispatch and service orchestration are channel-oriented; therefore, only specific services within channels are involved in the computation. Since the channels are independent, they can operate in parallel, thus saving a lot of time under PI. In the No-PI case, all services within the edge-cloud system involve in the computation of solving the problem (2)

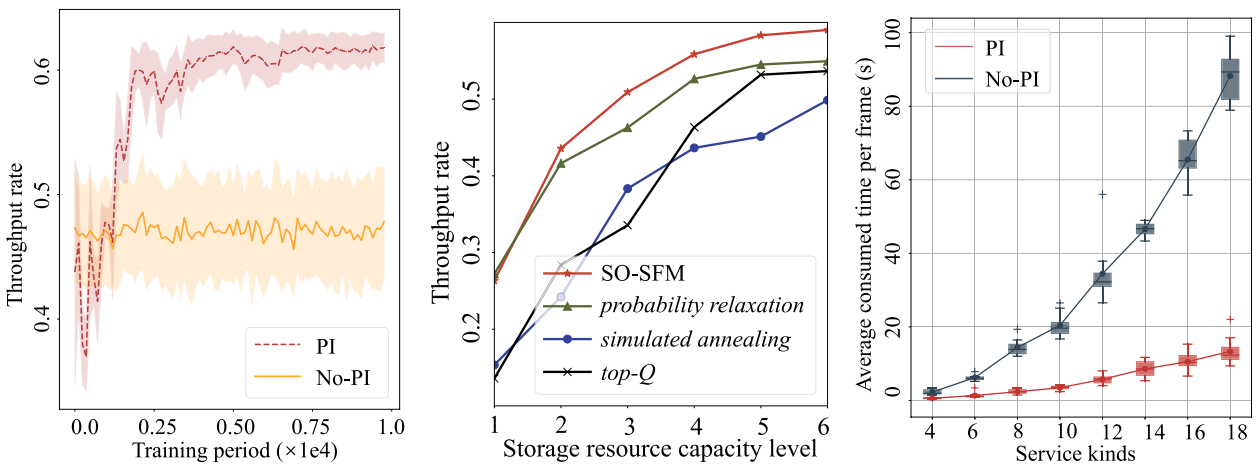

Fig. 4 Performance of request dispatch in the edge-cloud system: **a**) comparison of throughput rate under PI or non-PI; **b**) comparison of throughput rate of different service orchestration algorithms; **c**) comparison of average consumed time per frame

and executing algorithm SO-SFM to necessarily consume more computational time, which is showned by Fig. 4(c).

We find it feasible to implement dynamic SLA customization and guarantee SLAs through optimization in an edge-cloud system through the above experimental analysis. Moreover, there is considerable improvement in the performance metrics we are concerned about, such as throughput rate and average consumed computational time.

Discussion

Performance isolation achieved through *SLA-ORECS* still faces challenges, such as the need to deploy DRL agents on edge nodes to store training and test data, train models, and update parameters. However, all of the above challenges are inevitable in the training process of machine learning applications. For large-scale machine learning applications, training and testing models can take a lot of time. Still, fortunately, there are already many proven distributed and efficient training solutions to draw from.

On the other hand, there are many opportunities for *SLA-ORECS*, such as 1) service orchestration and resource customization are obtained simultaneously using DRL decisions so that training results can directly replace algorithmic predictions; First, service orchestration itself is a very complex task, which needs to consider the distribution of requests, the heterogeneity of resources, and the heterogeneity of edge nodes, and there are limitations in using traditional heuristic algorithms for optimization. In addition, there is a strong correlation between service orchestration and resource customization, and more services for orchestrating means more space for resource customization. Due to the above considerations, using deep reinforcement learning decisions to obtain both service orchestration and resource customization is expected to be the solution from the present point of view. 2) in addition to the performance metrics we mentioned in our experiments, users can feed their more focused metrics to the DRL reward function to get more satisfactory results through training. In our experiments, the metric we focus on is the number of demands. Therefore, we choose the throughput rate as the reward function in deep reinforcement learning. If the user is more concerned about the latency, it can be included in the reward function. Even if the user is more concerned about the combined effect, other metrics such as throughput and latency can be included in the reward function. 3) the time scale relationship between *SLA-ORECS*'s implementation and request dispatch can dynamically change to better suit reality at the time; For example, if the current resource customization and service orchestration results match the current request better, then the timescale can be dynamically adjusted so that a large timescale contains more small timescales. Accordingly, if the current resource customization and service orchestration results do not match the current request well, then the timescale can be dynamically adjusted so that a large timescale contains fewer small timescales. 4) alternatively, multi-agent decisions are used instead of single-agent decisions, where multiple agents interact with each other to decide on the allocation of resources for maximizing the utility of the edge-cloud system. In our experiments, we use single-agent decisions. However, in our subsequent work, we will implement the mechanism of *SLA-ORECS* in a multi-agent manner based on the above points. Whether challenges or opportunities, these issues remain open questions and areas for consideration in future.

Conclusions

We have discussed the necessity for dynamic SLA customization and SLA guarantee in an edge-cloud system and mathematically modeled the resource customization, service orchestration, and request dispatch. We have proposed *SLA-ORECS*, a scheduling framework that combines artificial intelligence algorithms with optimization algorithms to achieve dynamic performance isolation and SLA guarantee in edge-cloud systems. Experimental results corroborate the effectiveness of *SLA-ORECS* and show its promising performance in system throughput and average time consumption. Our future work will deploy and enhance *SLA-ORECS* in Kubernetes-based multi-cluster edge-cloud systems, focusing on improving their scheduling timeliness.

Code availability
Not applicable.

Authors' contributions
Conceptualization, S.Lan. and B.T.; methodology, Z.D. and Shi.Chen.; investigation, Y.L.; resources, Shan.Chen.; writing—original draft preparation, Z.D.; writing—review and editing, S.Lu.; project administration, S.Lan. All authors have read and agreed to the published version of the manuscript.

Funding
No funds have been received from any agency for this research.

Availability of data and materials
Not applicable.

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare no competing interests.

Received: 7 November 2023 Accepted: 30 November 2023
Published online: 15 January 2024

Publisher's Note

References

1. Li Y, Wu Y, Dai M, Lin B, Jia W, Shen X (2022) Hybrid NOMA-FDMA assisted dual computation offloading: a latency minimization approach. IEEE Trans Netw Sci Eng 9(5):3345–3360
2. Guo K, Yang M, Zhang Y, Cao J (2022) Joint computation offloading and bandwidth assignment in cloud-assisted edge computing. IEEE Trans Cloud Comput 10(1):451–460
3. Kamran K, Yeh E, Ma Q (2022) Deco: Joint computation scheduling, caching, and communication in data-intensive computing networks. IEEE/ACM Trans Netw 30(3):1058–1072
4. Santos Bernardino J, Correia N (2023) Automated application deployment on multi-access edge computing: A survey. IEEE. 11:89393–89408. https://doi.org/10.1109/ACCESS.2023.3307023.
5. Yin B, Cheng Y, Cai LX, Cao X (2017) Online sla-aware multi-resource allocation for deadline sensitive jobs in edge-clouds. In: GLOBECOM 2017-2017 IEEE Global Communications Conference, IEEE, pp 1–6
6. Katsalis K, Papaioannou TG, Nikaein N, Tassiulas L (2016) SLA-driven VM scheduling in mobile edge computing. In: 2016 IEEE 9th International Conference on Cloud Computing (CLOUD), IEEE, pp 750–757
7. Liu Q, Choi N, Han T (2021) Constraint-aware deep reinforcement learning for end-to-end resource orchestration in mobile networks. arXiv preprint arXiv:2110.04320
8. Liu Q, Han T, Moges E (2020) Edgeslice: Slicing wireless edge computing network with decentralized deep reinforcement learning. In: 2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS), IEEE, pp 234–244
9. Chen X, Zhao Z, Wu C, Bennis M, Liu H, Ji Y, Zhang H (2019) Multi-tenant cross-slice resource orchestration: a deep reinforcement learning approach. IEEE J Sel Areas Commun 37(10):2377–2392
10. Alsaffar AA, Hung PP, Huh EN (2017) An architecture of thin client-edge computing collaboration for data distribution and resource allocation in cloud. Int Arab J Inf Technol 14(6):842–850
11. Liu Y, Yu H, Xie S, Zhang Y (2019) Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks. IEEE Trans Veh Technol 68(11):11158–11168
12. Zhao N, Liang YC, Niyato D, Pei Y, Wu M, Jiang Y (2019) Deep reinforcement learning for user association and resource allocation in heterogeneous cellular networks. IEEE Trans Wirel Commun 18(11):5141–5152
13. Chen J, Wei Z, Li S, Cao B (2020) Artificial intelligence aided joint bit rate selection and radio resource allocation for adaptive video streaming over f-rans. IEEE Wirel Commun 27(2):36–43
14. Zhang Z, Chen H, Hua M, Li C, Huang Y, Yang L (2019) Double coded caching in ultra dense networks: caching and multicast scheduling via deep reinforcement learning. IEEE Trans Commun 68(2):1071–1086
15. Peng H, Shen X (2020) Deep reinforcement learning based resource management for multi-access edge computing in vehicular networks. IEEE Trans Netw Sci Eng 7(4):2416–2428
16. Farhadi V, Mehmeti F, He T, Porta TFL, Khamfroush H, Wang S, Chan KS, Poularakis K (2021) Service placement and request scheduling for dataintensive applications in edge clouds. IEEE/ACM Transactions on Networking 29(2):779–792 https://doi.org/10.1109/TNET.2020.3048613
17. Wang Y, He H, Tan X (2020) Truly proximal policy optimization. In: Proceedings of The 35th Uncertainty in Artificial Intelligence Conference, pp. 113–122. PMLR
18. Krause A, Golovin D (2014) Submodular function maximization. Tractability 3:71–104
19. Aliababa-clusterdata. https://github.com/alibaba/clusterdata. Accessed 10 Oct 2021
20. Ppo-hyperparameter-settings. https://github.com/quantumiracle/Popular-RL-Algorithms/blob/master/ppo_gae_discrete.py. Accessed 31 Sep 2022
21. Bertsimas D, Tsitsiklis J (1993) Simulated annealing. Stat Sci 8(1):10–15